ON DECISION TREE INDUCTION FOR KNOWLEDGE DISCOVERY IN VERY LARGE DATABASES

By

JOSE RONALD ARGUELLO VENEGAS

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1996

To my sons

ACKNOWLEDGMENTS

I will be always grateful to Dr Sharma Chakravarthy for holding faith in me despite our innumerable discussions and for guiding me through my Ph D program His advice has provided me with an intellectual guide in taking on this subject and pursuing it to its limits I have enjoyed and benefited from his acute analyses and ideas

My thanks go to the other members of my supervisory committee for accepting to spare their valuable time and effort to help me with my work Their comments and suggestions have been very valuable

I am thankful to the Database Research Center for all resources and for providing this wonderful opportunity, and to the Computer & Information Sciences & Engineering (CISE) department for making it possible

I am thankful to the Conicit of Costa Rica and all the people from the Department of Human Resources specially to Elvia Araya for all her hard work and help during my studies

My thanks to the University of Costa Rica and the people of the International Affairs Office and its chairman Dr Manuel Murillo for his continued communication and support

I thank my wife Elizabeth who had to adapt to a new culture learn a new language, take care of our son, and our home I thank her for her belief in me and for her patience

I must also thank my sons Ronald Jose Pablo and Roy Antonio who suffered the same way I thank them for their patience in letting their father leave to pursue a program which meant that for x years he became just a computer screen to communicate with I must thank Roy especially his sickness gave me a focus when I needed it most To Sebastian–the youngest one–who never understood why his father was always sitting at the computer and why he wasn t allowed to and who occasionally opted to turn off the computer without my noticing

My thanks go to my mother Emerita and father Filadelfo for their continued support, and to my sisters and brother whom I missed during my stay in USA and who have had to deal with all the troubles I have created with my leave My special thanks to Giselle and Sonia for supporting me and for taking on my matters while I was away

I must also thank my brother-in law Manuel Lopez, my friends and colleagues Raul Alvarado and Ileana Alpizar for giving me the initial support Similarly, I must thank Manuel and Ligia Bermudez who helped us in innumerable different ways during our stay in Gainesville.

Thanks too, to Susan and Janet Lewis who helped us with personal matters and with reviewing the manuscript, however any errors that still remain are my unique responsibility

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abstract of Dissertation Presented to the Supervisory Committee
in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

ON DECISION TREE INDUCTION FOR KNOWLEDGE DISCOVERY IN VERY LARGE DATABASES

By

JOSE RONALD ARGUELLO VENEGAS

August 1996

Chairman Dr Sharma Chakravarthy
Major Department Computer and Information Sciences and Engineering

Knowledge Discovery in Databases is the process of extracting new patterns from existing data. Decision Tree Induction is the process of creating decision trees from samples of data and validating them for the whole data base. The approach taken in this project uses decision trees not just for solving the classification problem in Knowledge Discovery but for forming association rules from them which are in effect new and explicit knowledge. Several performance problems need to be addressed for using a decision tree approach to large scale databases. I offer a new criterion which is better suited to decision tree construction and its mapping to association rules. The emphasis is on efficient, incremental and parallel algorithms as effective ways to deal with large amount of data. Comparisons with existent systems are shown to illustrate the applicability of the solution described in this dissertation to the problem of finding rules (knowledge discovery) and classifying data in very large databases

CHAPTER 1
INTRODUCTION

1.1 Motivation

Knowledge Discovery in the context of large databases is an area of growing interest [35] [12] [24] [1] [3] [20] [37] [2] [21]. Knowledge Discovery or Data Mining is the process of making explicit patterns that are implicit in the data been analyzed. These patterns represent knowledge embedded in the data under consideration. Discovering them, i.e., making them explicit, is the subject of every data mining system.

However there is not general agreement on which type of patterns must be discovered, the general consensus is to express patterns as if-then rules that are satisfied by part or the whole data, such as "if the temperature is higher then 100 degrees then color is red", "if the customer spends more than $50 then a gift is given" and "if region is northwest then precipitation is high". Additionally, to be of practical interest it is important to know the probabilities /confidences associated with each of these new patterns.

Data mining requires the convergence of several fields: data bases, statistics, machine learning and information theory. How they interact is still under study. Toward that one, Frawley, Shapiro and Mathews [12] introduced a model for knowledge discovery depicted in figure 1.1.

Their model summarizes the primary functions a system must perform for data mining:

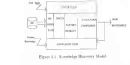


Figure 1.1: Knowledge Discovery Model

- *Database Interface:* Most recent work on data mining can be called file mining [22] because it lacks this primary component: a way to access existent databases by an interface language. See, for example, Han et al. [21].

- *Focus:* This component is the ability of the system to select relevant data and avoid processing the entire data set (which is typically very large).

- *Pattern Extraction:* This part is the specific way to extract, manipulate and represent specific patterns from the database. A mechanism able to search for specific patterns like if-then rules, semantic nets or decision trees.

- *Evaluation Component:* This component is the actual method used to filter or discard rules and keep those more meaningful ones for output or later processing in the knowledge database (a data base of rules and domain knowledge in form of rules or in the specific pattern representation of the system).

- *The Controller:* This component consists of the part of the system that interfaces with the user and guides the other components.

1.2 Classification and Rule Extraction

Common to all data mining systems are two primary functions: classification and rule representation. Classification is useful as a way to group data and focus the data analysis process. Rule representation affects the expressive power of the extracted rules (linguistic bias), the amount of knowledge discovered and the evaluation process. Decision Tree based algorithms have been proved to be good classifiers in the machine learning field. Induction by decision trees is perhaps one of the best known methods in machine learning despite of its lack of application to large data bases.

Their use in inductive inference based systems for small data sets has been very well investigated and documented. In addition to their ability to classify new data, decision trees can be used in a variety of ways in knowledge discovery:

- They can represent a functional dependency and the number of tuples that satisfy the dependency in a populated database.
- A decision tree derived from the data can capture potential rules present in the data and can therefore guide the user in the process of rule discovery.
- Since each attribute of the database can induce a partition according to its range, the decision tree associated with (or derived from) this partition determines the respective association rules for the attribute.
- A decision tree can be pruned and transformed to reduce its size, improve its classification accuracy and to represent meaningful and general rules.

I have found that decision trees can function correctly and efficiently only if we provide those functions and capabilities described for a Knowledge Discovery model to any decision tree based system

I am proposing the use of decision trees not just for solving the classification problem in knowledge discovery, but for extracting rules implicitly represented inside the data.

The use of decision trees in very large databases and in distributed ones requires a compromise so they can operate efficiently in such an environment while preserving the accuracy and quality of the knowledge discovered There is also a need for designing a suitable interface and the data mining operators needed to retrieve data from the database. However my concern is with classification and rule representation with decision trees in large databases

CHAPTER 2
ALGORITHMS FOR KNOWLEDGE DISCOVERY IN DATABASES

2.1 SLIQ, A Fast Scalable Classifier for Data Mining

SLIQ was developed by M. Mehta, R. Agrawal an J. Rissanen at the IBM's Almaden Research Center [26]. The objective of SLIQ is to solve the classification problem for data mining using scalable techniques. It is a decision tree extraction system for very large data sets that creates inverted lists for the attributes and uses splitting /subsetting of attributes as a criterion for attribute selection. Additionally, tree pruning is used to improve the accuracy of the resulting tree. Initially, inverted lists for each attribute are created. Then, attribute selection is done by pre-sorting numerical attributes and by finding the best interval split with the gini index [6]. A fast algorithm for selecting the best subset for categorical attributes is used.

2.1.1 The Algorithm

Data structures

Attribute lists. A set of lists. One for each attribute. Each list contains the attribute value and a tuple index.

Class list. An ordered list of class values for each tuple and the corresponding decision node associated (A decision tree partitions the data and every tuple is associated to the path of nodes from the root to the leaf node. Initially all tuples are associated with a single leaf node).

Decision Tree A binary decision tree In each node it keeps the node number, the decision attribute the decision value and the class histogram (Class counts for every class value to the left and right of the decision value)

```
Sliq ():
S0-     Read database and create a separated list for
          each attribute and tuple index (Attribute list)
          For every class value associate an initial node
          n1 (the leaf) and create the list with class values
          and nodes (Class List)
          Initialize the class histogram
S1      Presorting:
          Sort all attribute lists by attribute value
S2-     Partition(data S)
S2.0-       If (all tuples are in the same class) return
S2 1        Evaluate Splits
              For each attribute A do
              For each value v do
                  Use the index to get class value and leaf node L
                  Update the class histogram
                  if A is a numeric attribute then
                      Compute splitting index for A <= v for leaf L
              if A is a categorical attribute
              For each leaf of the tree do
                  Find subset of A with best split
S2.2        Use best split found to partition the actual data
              into two sets S1 and S2,
S2.3        Update class list
              For each attribute A used in the split do
              For each value v do
                  Find the entry in the class list e
                  Find the new class c to which v belongs
                  by applying the splitting test in node referenced by e
                  Update the class label for e to c
                  Update the node referenced in e to the child
                  corresponding to the class c
```

S2.4 Partition(S1)

S2.5 Partition(S2)

2.1.2 Merits

Sliq performs similar to or better than others classifiers for small data sets and the classification time is almost linear for large data sets [26]. This is the first case where more than 100000 instances were used (until 10 million). The pruning method influences significantly its performance. Important contributions are scalability and breadth first growth as well as subsetting for categorical attributes and pruning using the Minimum Description Length principle. The use of synthetic databases with more than 100000 cases prove the scalability of SLIQ.

2.1.3 Limitations

SLiq derives a decision tree that correctly classifies the training set and gets high accuracy for the whole set, but it is not incremental. It is designed to classify the training data set but without using reduction or learning capabilities i.e. it processes the entire database to get the final tree. Also, it does not use parallelism or distribute to decision tree generation. SLIQ makes at most two complete passes over the data for each level of the decision tree [26, pp 20]. The attribute evaluation criteria requires pre-sorting and evaluation of all possible splits for each attribute making this phase a time consuming task.

2.1.4 Summary of SLIQ Features

In summary, SLIQ is similar to standard decision tree algorithms like CART and C4.5 described in [26]. SLIQ requires two times more space than the original database (since columns are kept as separated list with indices attached) when numerical attributes are present. When symbolical or categorical attributes (strings) are used, the amount of space required is increased by the size of the indexes to the data base The algorithm is faster is the sense that it uses just one pass for every level of the decision tree, but the actual volume of the inverted lists is almost two times the initial data base volume, increasing the number of I/O accesses This is particularly significant in a very large data base environment

2.2 Systems that Extract Rules from Databases

2.2.1 Systems for Extracting Association Rules

Several algorithms have been proposed to extract association rules from data [1], [19], [3], [37], [20], [2] Most of these systems are based on the original algorithm proposed by R Agrawal, called the *Apriori* algorithm [1]

2.2.2 The Apriori Algorithm

The basic algorithm is summarized below

```
The Apriori algorithm
s1 L(1) =  frequent 1 itemsets
s2 k = 2, // k is the pass number (<= number of attributes(columns))
s3 while  ( L(k 1) is not empty ) do
s4       C(k) = New candidates of size k generated from L(k 1)
s5       for all  transactions t in data base do
s6       For all  c(k) in C(k) do  c(k) count++
s7       L(k)= all candidates in C(k) with minimum support
s8       k++
s9 end
s10 Answer= Union of L(k)  for all k
```

Complexity

if A is the number of attributes, then Step s3 is done A times in the worst case. In step s5, the whole data base is traversed. So we have, at most, A number of passes over the data base.

Step s4 is compute intensive, but the main concern is the number of passes over the data base and therefore the number of I/O's incurred for that purpose.

An improvement to the previous algorithm, called the *AprioriTid* algorithm proposed also proposed by Agrawal and Srikant [3] suggested that a data structure be used to discard transactions in step s5. If a transaction does not contain any large itemsets in the current pass, that transaction is no longer considered in subsequent passes.

2.2.3 Description of Parallel Approaches

The goal of Parallel systems is to extract association rules by using parallel processing techniques.

Approach

This is achieved by parallelizing the serial algorithm - the Apriori algorithm - which counts the support of each itemsets and finds rules based on the frequent itemsets. The support is the percentage of transactions (tuples) that contains the itemset. The frequent itemsets are those with a minimum user specified support. There are three possible algorithms:

1. The *count distribution* algorithm - in which basically each processor counts the support locally and distributes this to all other processors.
2. The *data distribution* algorithm in which the total memory of the system is exploited - a disadvantage of the previous one. This algorithm counts locally the mutually

exclusive candidates (viable itemsets) and then the local data must be broadcast to all processors

3 The last algorithm (the *candidate distribution*) tries to make each processor work independently since in the previous algorithms each processor locally extracts the candidate sets Synchronization is needed at the end of every pass. The idea is that each processor can generate unique candidate sets independent of other processors dividing appropriately the frequent itemsets However, not all dependencies are eliminated

Additionally, a parallel algorithm is presented to generate rules from frequent itemsets

Merits of the three approaches

The three algorithms give clear ideas of how to parallelize the serial algorithm The use synthetic data to evaluate the algorithms and their performance, scaleup , sizeup and speedup primarily by the count distribution algorithm

Limitations

The Count and Distribution algorithms perform equivalently to the serial algorithm The Data distribution requires fewer passes but its performance is worse than the others mainly because half of the execution time is spent in communication For scale up where databases were increased proportionally to the number of processors, the Count distribution performs very well and almost constant accordingly to the number of processor involved For sizeup increasing the size of the database but keeping the number of processors constant the Count and Candidate algorithms show sublinear performance For speedup- keeping the database constant and adding more processors, the Count distribution is better and performs almost linear up to 16 processors

The number of passes over the data is the same for all algorithms except the data distribution algorithm. The number of passes is proportional to the transaction length (since those are binary values and each represents an attribute-value, we may say that the passes are proportional to the number of attributes in the relation).

In the above approaches the whole database is processed as no learning algorithms are involved.

2.2.4 The Partition Algorithm for Deriving Association Rules

Another algorithm called the *Partition algorithm* introduced by Savasere et al. [37], which claims to need two passes over the data.

Basically the algorithm avoids passing over the data in step ab of the Apriori algorithm. Instead of reading the data base again to count the support of the Candidate sets, it keeps the transaction list of each set. Counting is done by taking the intersection of those lists.

The algorithm is called Partition since it can apply the modified Apriori algorithm to parts of the database; then all local large itemsets are joined to get the final large itemsets. In order to merge all local large itemsets an additional pass is necessary.

The performance results show that for lower minimum support values (less than 1%) the Partition algorithm outperforms the Apriori Algorithm. The reason for this (our opinion) is that lowering the support, the transaction lists of each itemset are shorter and can be kept in memory without additional disk accesses. They show the results for 100K transactions or more, so a 1% or lower support means no more than 1000 numbers that can easily be kept in memory. It seems that the authors replace data base passes with transaction lists passes since they kept every data base part in memory and therefore the savings are for small values of support.

CHAPTER 3
DECISION TREE CONSTRUCTION

3.1 The Tree Construction Algorithms

The basic algorithm for decision tree induction was introduced by J R Quinlan [27] [31] Incremental solutions based on tree restructuring techniques were introduced by Schlimmer and Utgoff [38] [44] Those algorithms requires one pass over previously seen data per level in the worst case, as does Van de Velde's incremental algorithm, IDL, based on topologically minimal trees [10] This section describes the algorithms to build the tree for a sample of data, either directly or incrementally

3.2 The Centralized Decision Tree Induction Algorithm

Quinlan's traditional algorithm for decision tree induction [27, pp 469] was as follows

```
(s1)     Select a random subset of the given instances (the window)
(s2)     Repeat
(s2 1)       Build the decision tree to explain
             the current window
(s2 2)       Find the exceptions of this decision
             tree for the remaining instances
(s2 3)       Form a new window with the current
             window plus the exceptions to the
             decision tree generated from it
         until there are no exceptions
```

Step 2 1 is called Decision Tree Derivation and step 2 2 is called Decision Tree Testing. Step 2.3 is the major drawback in the above algorithm since it forces the process to pass over all the training data (the window) again, and therefore the algorithm is not incremental. The algorithm presumes that none of the instances are stored within the decision tree thus preventing the algorithm for being incremental, and also assumes that no additional information is needed in each node besides the decision data.

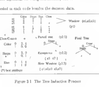


Figure 3 1 The Tree Induction Process

The Decision Tree Derivation (step 2 1) proceeds in two stages – a selection stage followed by a partition stage

Derivation Algorithm

```
(s2.1.0) If all instances are of the same class
         the tree is a leaf with value equal to the
         class, so no further passes are required
(s2.1 1) Select the best attribute (the root)
         according to a criterion   usually statistic
(s2.1 2) Split the set of instances according to each
         value of the root attribute
(s2.1.3) Derive the decision subtree for each subset
         of instances
```

Steps s2 1 1 and s2.1 2 of this algorithm, the selection and partition steps respectively, each require one pass over the data set. Selection steps usually count the relative frequency in the data set of every attribute-value with the class value (Class counts) which are then used statistically to compute the best attribute (the root). The partition steps distribute the data across the different branches of the root attribute. Thus, the algorithm in general requires two passes over the data per level of the decision tree in the worst case.

3.3 The Selection Criteria

The basic criterion generally used for attribute selection is the information gain criterion suggested by Quinlan [27]. The information gain criterion minimizes the average attribute entropy:

$$E(A) = \sum_{a \in \mathcal{D}(A)} P(A = a) H_n(A = a) \tag{3 1}$$

where P(A=a) is the relative probability of $A = a$ and for a set of n potential classes, $H_n(A = a)$ is the entropy for the set defined for all tuples in which $A = a$

$$H_n(A = a) = -\sum_{i}^{n} p_i(A = a) \log(p_i(A = a)) \tag{3 2}$$

where $p_i(A = a)$ is the relative probability of being in class i when $A = a$

A different form to express the criterion for attribute selection which instead of minimizing the entropy, maximizes the certainty and is given by

$$E(A) = \sum_{a \in \mathcal{D}(A)} P(A = a) CH_n(A = a) \tag{3 3}$$

$$CH_n(A = a) = 1 - \frac{H_n(A = a)}{\log n} \tag{3 4}$$

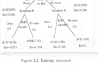


Figure 3.2. Entropy measures

3.4 The Incremental Algorithms

As mentioned above the incremental algorithm originally devised by Schlimmer [58] and Utgoff [44] avoids passing unnecessarily over previously seen instances. To achieve this, it is necessary to keep all Class counts in every node of the decision tree and it is also necessary to create a mechanism to access previous cases at all leaves of the decision tree for restructuring the tree during the incremental phase. This mechanism is omitted in most implementations since it is assumed that all instances (data base) will be kept memory resident. All previous algorithms start with an empty tree and gradually modify its structure according to the input instances. For every new instance, there is a potential cost of one pass per level over all seen instances.

This cost is half of the cost of directly deriving a tree for traditional algorithms. Hence the importance of the incremental version.

The algorithm below will derive the tree for a part of the data base and then update it incrementally (the updating phase) using one instance at a time.

Incremental Induction Algorithm.

```
(s0)    Select a random subset of the data base (the window)
(s1)    Build the decision tree to explain the
          current window (the Tree)  keep Class
          counts in every node
(s3)    While  there are exceptions  do
(s3 1)      Find a exception of the decision tree
              in the remaining instances
(s3.2)      Update the decision tree Class counts
              per node using this exception
(s3 3)      Reorganize (Tree)  See below
          done
```

Incremental algorithms usually start with a random subset of one element The algorithm above doesn t preclude this possibility

3.4.1 Tree Reorganization

Tree reorganization is the key for incremental algorithms including algorithms which are not based on statistics over the input instances [18] This technique is essential to avoid traversing the whole data base again when dealing with very large databases Hopefully tree reorganization will require just a small part of the database when the tree is restructured.

The reorganization part depends on the relative representation suited for the algorithm Utgoff maps every attribute-value pair to a new boolean attribute [45] Thus he assumes all trees are binary trees

Tree reorganization algorithms restructure the tree when a better attribute is detected (or inherited in the case of a subtree) The basic idea is to force all subtrees to keep the same root (the best attribute) and then apply a transformation rule to exchange the actual root of the tree with each subtree (See figure 3 3 and algorithm below) In this way, some subtrees are pruned when all subtree branches lead to the same class value

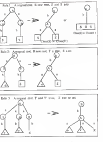

Rule 1 A original root, B new root, S and S' sets
a
b
S
S'
or
S U S'
Class(S) = Class(S')
Rule 2: A original root, B new root, T a tree, S a set
T
S
Rule 3 A original root, T and T' trees, X tree or set
X
T'

Figure 3 3 Transformation rules

The ID5R pull up algorithm reorganizes the decision tree in the way just mentioned [14]. If a tree is just a *leaf* (a set of instances) the pull up algorithm assumes the respective attribute as the root of the decision tree starting on that leaf. Then the leaf is *expanded*, i e , the decision tree is built

The ID5R pull up algorithm

(s1) If the attribute A to be pulled up is at the root, then stop

(s2) Otherwise

(s2 1) Recursively pull the attribute A to the root of each immediat[illegible]

(s2 2) Transpose the tree, resulting in a new tree with A at at the root, and the old root attribute at the root of each immediate subtree.

Note that in step s2 2 the transformation rules of figure 3 3 must be applied to [illegible] the transposed tree.

Van de Velde s algorithm IDL uses the same pull up technique for reorganisation as ID5R [10]. IDL differs from others in that it uses a topological criterion called *topological relevance gain*, based on the tree structure to select the actual root attributes for the subtrees Van de Velde shows how his algorithm is able to discover concepts like the tree shown in figure 3 4 while traditional algorithms fail to discover this tree

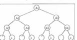


Figure 3.4 A tree for the 6 multiplexer

Basically, the topological relevance ($TR_m(A,e)$) criterion measures the number of occurrences of a given attribute A for a given example e when this is used to traverse the tree starting from any leaf of the example class all the way up until the node m if it is possible It depends uniquely on the actual tree structure and the given example Thus, given nodes m and s in the classification path of an example e with s the immediate son of m, the *topological relevance gain* for attribute A is

$$TRG_m(A, e) = \frac{TR_m(A, e) - TR_t(A, e)}{TR_m(A, e)} \tag{3.5}$$

When compared to its predecessor ID5R, IDL saves computations costs in terms of class counts, criteria computations expansions of sets pruning and transformations while keeps better or similar accuracy

More recently Utgoff has implemented the ITI algorithm which is a direct descendant of ID5R and uses reorganization like techniques in a similar way [45]

3.5 Other Approaches

SLIQ , a fast scalable classifier for Data Mining (described in chapter 2, was designed to solve the classification problem for Knowledge Discovery [26] Conceptually, the SLIQ system uses the same algorithm where the selection criterion is the gini index a criterion that splits the range of numerical attributes in two parts It also uses set splitting for categorical attributes The gini index for a set S containing n classes is

$$G(S) = 1 - \sum p_j^2 \tag{3.6}$$

where p_j is the relative frequency of class j and then the attribute measure is

$$G(A = u) = P(A \leq u) * G(A \leq u) + P(A > u) * G(A > u) \tag{3.7}$$

where $A \leq u$ or $A > u$ represents the set of tuples that satisfies the relation

Thus, SLIQ represents [illegible] binary decision tree. In order to make the system scalable, most of the data are handled off line with inverted lists for all attributes and a special Class list that maps the instances to the nodes of the decision tree. This Class list is maintained in main memory. The system incorporates tree pruning using the *Minimum Description length principle*. Mehta et al show that the system achieves similar or better performance than IND-Cart and IND-C4 (ID3 descendants) for different data sets, especially for larger data sets (20000 to 60000). They also show that for synthetic data bases of millions of cases SLIQ achieves almost linear performance on the number of tuples and number of attributes.

3.6 Applicability to Large Databases

Decision trees for Knowledge Discovery in large databases can be applied in two related areas: classification and rule extraction. Although, rule extraction from decision trees is not new [4], [32], applications of decision trees have been oriented to the classification problem. Nevertheless, the decision tree algorithms mentioned have the following problems when used for large databases:

1. Their study has been primarily done on small data sets (from hundreds of cases to a few thousands). It is only recently that researchers are interested in its application to large data sets. See [26].

2. Incremental issues in decision tree induction have not been studied in the context of large data bases. In general, the cumulative cost of the pure *incremental* algorithm -one instance at a time- will preclude its use over a direct derivation algorithm over the data base.

3 There has not been any related work on the mapping between decision trees and association rules for data mining. Levels of support and confidence in decision trees and its relationship to the attribute selection criterion is not mentioned in the references.

4 Traditional algorithms assume that instances and class counts are kept in memory regardless of the number of tuples involved. Sliq is the other extreme case where all information is off line.

5 Recent algorithms like ITI and Sliq represent or transform the attributes to binary ones. This is adequate when the decision tree is just a classification tool but it appears inadequate when rules have to be extracted and a close resemblance of the original attributes is a must for the end user.

6 Theoretical analyses so far have been oriented to the computation of class counts, the criteria and transformations, but never to the number of passes over the data since it was supposed to be memory resident.

7 Induction techniques for large or distributed databases have not been studied.

CHAPTER 4
EXTENSIONS OF DECISION TREE CONSTRUCTION ALGORITHMS

4.1 Problems in Classical Tree Construction Algorithms

The basic algorithm for decision tree induction introduced by J. R. Quinlan had two major drawbacks for its use in very large databases: it was not incremental and it required in the worst case, two passes over the entire *data per level* to build the decision tree [31].

The incremental solution based on tree restructuring techniques [38] [44] requires one pass over previously seen data *per level* in the worst case, as does Van de Velde's incremental algorithm *IDL* based on topologically minimal trees [10]. This makes the utility of the incremental version more attractive for large databases. However, the incremental version requires keeping the data "inside" the decision tree structure [45] in main memory and hence it is likely to have a high cumulative cost [26], which partially precludes its use for large databases. This section describes a *one-pass per level* worst case algorithm to build the tree for a sample of data, which makes it equal to or even better than building the tree incrementally. In either case, the expected number of nodes of a decision tree in very large databases requires a mechanism to store part of the tree in external memory. Since the size of large databases precludes keeping several copies of the data, data can be incorporated into the tree leaves using indices to the main database or using the tree as a way to fragment the database.

4.2 Extensions to the Centralized Decision Tree Induction Algorithm

4.2.1 Minimizing the Number of Passes over the Data

To minimize the number of passes over the data base, the split step and the selection of the next step need to be combined in one pass. The trick is to use each case (tuple) to update the Class counts of the corresponding subtree (or subset) and to create the data subset simultaneously. Thus, in the next selection step, there will be no need for an additional pass over the subsets for every subtree in the next level. Then, even in the worst case, we will need only one pass per level over the data base.

The first step of the derivation must proceed like this.

Derivation Revisited (Initial step)

(s2.1.0) **If** all instances are of the same class
    the tree is a leaf with value equal to the
    class, so no further passes are required.

(s2.1.1) **Select** the best attribute (the root)
    according to a criterion, usually statistic

(s2.1.2) **Split** the set of instances according to each
    value of the root attribute
    Update Class Counts for every subtree with each instance

(s2.1.3) **Derive** the decision subtree for each subset
    of instances

Then, for each subtree:

Derivation Revisited

(s2.1.0) **If** all instances are of the same class,
    the tree is a leaf with value equal to the
    class, so no further passes are required

(s2.1.1) **Get** the best attribute (the root)
    according to a criterion, usually statistic

(s2.1.2) **Split** the set of instances according to each
    value of the root attribute

```
        Update Class Counts for every subtree with each instance
(s2 1 3) Derive the decision subtree for each subset
        of instances
```

Note that the initial step requires two passes to check the data. After that, the remaining steps require just one pass per level The selection step does not require a pass over the data since all Class Counts were computed previously

The merging of these two steps is not without cost Additional memory is required to keep all frequencies (Class counts) for every subtree If we keep all those frequencies in memory, then it is clear that the decision tree can be built for every subtree, without additional disk accesses Note that only the class counts for the last level are needed and that the number of counts maintained in main memory are fewer whenever the level (of the tree) is higher However, there can be thousands of leaves in a decision tree for a large data base Eventually a mechanism to keep the class counts outside of main memory is needed But even if this is done for every subtree, additional disk accesses will be incurred for constructing each subtree The number of additional disk accesses for reading class counts will in general be lower than the number of disk accesses required to read the whole subset A threshold mechanism to avoid incurring those overhead costs for small data sets can easily be implemented

4.2.2 The Selection Criteria

Since the entropy based criterion has several limitations [31], I am proposing an alternative determination criterion to measure the certainty of a decision for a given class given by

$$D_{n}(A = a) = 1 - \frac{1}{n-1} \sum_{i \neq j} \frac{p_i(A = a)}{p_j(A = a)} \qquad (4\ 1)$$

where $p_j(A = a) = max_i p_i(A = a)$

Then, the average certainty per attribute is given by:

$$E(A) = \sum_{a \in \mathcal{D}(A)} P(A = a) D_n(A = a) \tag{4.2}$$

Intuitively, the determination guesses the most probable class in a given data set based uniquely on their relative probabilities. See chapter 5 for more details on the criterion.

Both measures, certainty based entropy as described in chapter 3 and the determination being the base for tree derivation, allow us to study the inductive behavior of the decision tree algorithms.

4.2.3 Improving the Halting Criteria

The Tree Derivation Algorithm halts when all instances in the data set are from the same class (*step* 2.1.0). It is impractical to expect this since data can be inconsistent or incomplete in the sense that there are not enough attributes to correctly classify the data. Thus, a threshold criterion must be introduced to stop the process when the *set measure* is beyond a certain point. The set measure corresponds to the same statistic used to evaluate and select attributes in step 2.1.1.

Quinlan's algorithm assumes that if all data are not from the same class, the attribute selection step will improve the classification. The following case shows this is not necessarily true. Suppose we have two classes with a distribution of 90% for positives and 10 % for negatives. Assume that every attribute splits the set in two halves, each one with 45% positives and 5% negatives. The best selected attribute will be either of them; but the average measure will be the same since the relative distribution of classes in each leaf is

Table 4 1 Entropy conjecture

Set Distr		Part 1		Part 2		Part 3		Part 4	
+	90	45	45	45	45	60	30	80	10
	10	5	5	10	0	10	0	0	10
CH_n	0.53	0.53	0.53	0.32	1.00	0.41	1.00	1.00	0.00
Avg		0.53		0.62		0.59		0.80	
$D(\vec{p})$	0.89	0.89	0.89	0.78	1.00	0.83	1.00	1.00	0.00
Avg		0.89		0.88		0.88		0.80	

the same as it is in the original data set. The information expected criterion will give us 0.47 entropy (0.53 certainty) in both cases. As the result is equal to the set measure, no improvement has been made

Even though, the previous example is an extreme case, usually absent in practice, the algorithm must check for this condition. In general, the algorithm must check if the *average certainty* –equation 4 2– is below or equal to the *set certainty*

For the entropy measure, the following seems to be true

Conjecture 1 Let S the data set, A an attribute Then

$$E(A) = \sum_{a \in \mathcal{D}(A)} P(A = a) CH_n(A = a) \geq CH_n(S)$$

This says that the entropy based certainty will always be greater or equal to the *set entropy-based certainty* with any partition of the data set

Table 4 1 shows four partitions for a set with two classes with a distribution of 90% and 10% respectively. Note that for any partition there will be an average entropy based certainty higher than the original set entropy based certainty (column 1) [illegible] Determination measure the conjecture is not true. For example with 90% positive cases and 10% negative cases, (0 88 Determination) the partition in one set of 80% positive and 0% negative and another of 10% positive and 10% negative, does not lead to a better average determination (0 8(1) + 0 2(0) = 0 80) Note that the entropy (certainty) changes from 0 47 (0 53) to 0 20 (0 80)

This property of the Determination measure will allow us to prune the decision tree before it fits the data unnecessarily since there is no improvement in the measure On the contrary the entropy will continue choosing attributes (even if they are irrelevant to the classification) since entropy decreases (certainty increases) with every partition if the previous conjecture is true.

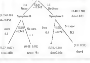


Figure 4 1 Determination measure

As an example consider the tree in figure 4 1 The same tree with entropy computed measures is shown in figure 3 2 Note that the certainty always increases when entropy is used This tree doesn't need to be built completely when determination is used The derived tree will be the tree depicted in figure 4 2. Note that both subtrees starting with root Symptom B were not needed since E(B) was always lower than the respective set determination (det) This coincides with the fact that the determination chooses the most general rule See chapter 5

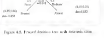


Figure 4.2. Pruned decision tree with determination

4.2.4 Pruning Using Confidence and Support

Related to the previous section but applicable in a different way is the mechanism to prune the tree The most general method is called the Minimum Description Length principle introduced by Quinlan [33] It has been successfully used in most of the actual systems [45], [25], [26] However it improves the accuracy and reduces the size of the decision tree the MDL principle is based on the future error and the cost of building the subtree pruned It is not related to implicit rules or to the user viewpoint In this sense, the pruning is artificial and of little or not interest to the user and the application

The confidence and support introduced here allow us to incorporate the end user and the meaning of the rules to be extracted as criteria to prune the tree The user can specify the thresholds for support and confidence When the subset cardinality in a leaf is below the minimum support or the confidence in the final classification is greater than a maximum confidence factor then the tree construction process must be stopped All potential rules will satisfy the requirements Note that, unlike the MDL principle, we don't care about the final error or the amount of work needed to build the tree Our goal is to meet the confidence and support thresholds.

Table 4.2 Pruning with the determination criterion

Test	Prune value	Tree Error	Tree Size	Tree Height	Tree Leaves	Tree Nodes
1	0.99	0.0005	2316462	7	2187	2688
2	0.95	0.0087	2835728	7	1914	2363
3	0.90	0.022	1677073	7	1565	1937
4	0.85	0.058	1174018	7	1094	1368

Similarly the attribute selection criterion gave us a good tool for tree pruning if we can predict the final outcome in terms of confidence or support. Entropy can not be used for this since there is no way to relate the entropy measure to the set confidence. In my opinion, this is the primary reason for the developing of pruning criteria such as the MDL principle.

Confidence and determination are related by

$\delta = D(p_1, p_2, ..., p_n) = \frac{nConf-1}{(n-1)Conf}$ and therefore $Conf = \frac{1}{n-(n-1)\delta}$

See chapter 5 for more details.

An artificial database with two classes, 5832 cases, 20 attributes plus class attribute was used to generate a decision tree with different determination levels (confidence levels). The results are shown in table 4.2.4. It can be observed that savings until one 50% on the size of the tree was achieved by pruning the tree with 85% determination (87% confidence) without sacrificing largely the error rate (no more than 5%).

4.3 Extensions to the Incremental Algorithms

As mentioned in chapter 3 the incremental algorithm originally devised by Utgoff [44], avoids passing unnecessarily over previously seen instances

With our one pass algorithm it is necessary to re-evaluate the incremental version - since the cost of both approaches is O(e) in general However, direct tree derivation is a *pessimistic* approach and assumes Nothing about the data Incremental algorithms are *optimistic* and they assume that the previous decision tree reflects the actual decision tree Using this information the practical performance of the incremental algorithms can be improved as compared to the direct (brute-force) approach I will discuss more thoroughly the re-organization approach used in incremental algorithms in a later section

In general, the cumulative cost of the pure *incremental* algorithm -one instance at a time- will preclude its use over a direct derivation algorithm over the data base The algorithm below will derive the tree for a part of the data base and then update it incrementally (the updating phase) using chunks of wrongly classified instances instead of one instance at a time

Partial Incremental Induction Algorithm

```
(s0)      Select a random subset of the data base
            (the window)
(s1)      Build the decision tree to explain the
            current window (the Tree) keep Class
            counts in every node
(s2)      Find the exceptions of this decision tree
            in the remaining instances
(s3)      **While** there are exceptions do
(s3 1)          Form a new window with a portion
                  of the exceptions to the decision tree
                  generated from it
(s3 2)          Update the decision tree Class counts
                  per node using the window
(s3 3)          Reorganize (Tree); See below
(s3 4)          Find the exceptions to this decision
```

```
        tree to the remaining instances
    done
```

4.3.1 Tree Reorganization Algorithms

As we discussed in chapter 3, tree reorganization algorithms restructure the tree when a better attribute is detected (or inherited in the case of a subtree). A more detailed algorithm for tree reorganization is given below. Again I have based the algorithm on the transformation rules in figure 3.3 of chapter 3.

The Reorganization Algorithm:

The reorganization procedure involves two parameters: the actual decision tree (Tree) and the new root attribute (NewRoot).

```
        Reorganize(Tree, NewRoot)
(s1)        If the NewRoot is null then
              NewRoot = better attribute for Tree
(s2)        If Tree is a leaf
(s2.1)          Create a new tree by splitting the
                  set according to the NewRoot
(s2.2)          Make Tree equal to this new Tree
(s2.3)          return
(s3)        otherwise (If Tree is not a leaf)
(s3.1)          If Tree.Root == NewRoot then
                  return
                  otherwise
(s3.2)          For each Subtree
(s3.2.1)          Reorganize(Subtree, NewRoot)
(s3.2.2)          Apply the transformation rule
```

```
(s3.2.3)        Update class counts for the
                subtree (now starting with
                previous root Tree Root)
(s3.3)      For each subtree STree,
                Reorganize(STree)
(s3.4)      return
```

In step s3, the best attribute is bubbled up until it reaches the root of the current decision tree. This is repeated for the next level of the decision tree until all subtrees hold the best attributes as roots or until they are just leaves. There is a potential for doing a pass over the data at the leaves for each level and therefore the algorithm requires one pass per level. However in practice, we expect that one attribute for the root candidate is already a subroot of a subtree and there is no need to reorganize the subtree. Thus, this algorithm will in general be better than the direct approach if the previous decision tree resembles the actual decision tree, which is likely since the tree was based on a representative subset (a percentage) of the actual data. See example in figure 4.3

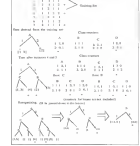

Instances
Training Set
Tree derived from the training set
Class counters
Tree after instances 4 and 5
Class counters
Root C
Root B
(counters for leaves are not included)
Reorganizing. (D is passed down to the leaves)

Figure 4.3 Tree Reorganization

4.4 Distributed Induction of Decision Trees

4.4.1 Distributed Subtree Derivation

The partitioning part of the Derivation algorithm (step s2) can easily be adapted to a multiprocess or a multiprocessor environment. Every data subset obtained in the partition is given to each available processor to continue with the tree derivation. Thus, the tree induction mechanism can easily be made in parallel. Additionally the subsets can be kept on secondary storage thereby allowing even larger sets to be used for induction with the restriction that the tree must be loaded into memory if the whole tree is needed for processing (for example for a centralized testing phase). However it is possible to design a mechanism to keep subtrees in secondary storage and loaded only when needed. The updating phase will proceed like any centralized algorithm. I term this the DSD algorithm.

The DSD algorithm

```
(s1)    Make a pass over the data set to select
          the attribute (the root)
(s2)    Split the data base (or create new index
          files) into as many data subsets as there
          are values of the root attribute
(s3)    Make each data subset available for other
          processes (saving one for self)
(s4)    While there are subsets
              apply the DSD to each subset
(s5)    If all data subtrees are ready, then
              make the decision tree attaching to
              each branch of the root the
              respective decision tree.
(s6)    End.
```

In step s2, the relative speed of every available processor can be taken into account or every subset will simply be distributed on a first-come first served basis. Similarly, in order to fully use the distributed capabilities of the system, a set will be available if its size is greater than a threshold set previously by the user.

The algorithm is useful when several processes or processors can cooperate to help in the decision tree derivation. It is assumed that they at least share a file system. For example the algorithm can be used when the decision tree does not fit in the memory available for each process or processor.

4.4.2 Distributed Tree Derivation

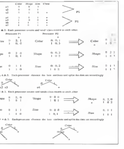
An alternative use of distributed processing capability in deriving decision trees is to assume that the training data is already distributed among processors (if not, a first pass can distribute the data equally among available processors). Thus, processors can interchange class counts on every attribute-value pair and then each one will arrive at the same conclusions on the selected attribute as a root. Then, as each data set will be partitioned accordingly, a new interchange of class counts will occur for each possible subtree, until the complete decision tree is derived for each processor. Communication is reduced to a minimum since data sets are not interchanged, just the attribute-value-class frequencies or Class counts (See Figure 4.4). This will be called the DTD algorithm.

For this algorithm, each processor has its own data set.

The DTD algorithm

```
(s1)   Make a pass over the local data set and
       create the Class Counts
(s2)   Send the Class Counts to every processor
(s3)   Receive the Class counts from each
       processor and summarize
```

```
(s4)    Select the best attribute (the root)
(s5)    If the tree is a leaf, return
            otherwise
                Split the local set according to the
                root values
(s6)    For every subset,
            recursively derive the tree
(s7)    Make the decision tree, attaching to each
            branch of the root the respective decision
            tree
```



Color
Shape
Size
Class
P1
P2
Step 1 & 2: Each processor counts and send class counts to each other.
Processor P1
Processor P2
Step 3, 4 & 5. Each processor chooses the best attribute and splits the data set accordingly
Step 1 & 2. Each processor counts and sends class counts to each other
Step 3, 4 & 5. Each processor chooses the best attribute and splits the data set accordingly

Figure 4.4 Distributed Tree Derivation

In the above algorithm, the Class Counts interchange among processors can be improved significantly if a processor is selected as a group coordinator and is in charge of the selection stage. This Coordinator will notify each processor the next root at every subtree. Each processor will send the Class Counts of its respective subset to the Coordinator. Thus, only one copy of the Class Counts will be transmitted. With the coordinator, the number of messages transmitted will change from $O(n^2)$ to $O(n)$, where n is the number of processors. The revised DTD algorithm is given below:

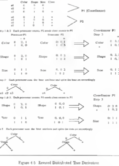
The Revised DTD algorithm:

```
(s1)    Make a pass over the local data set and
          create the Class Counts.
(s2)    Send the Class Counts to the Coordinator
(s3)    **If** Processor = Coordinator.
(s3.1)        Receive the Class counts from each
                processor and summarize.
(s3.2)        Select the best attribute (the root)
(s3.3)        Notify each processor of the root
                selected and next subset to process
(s4)    Wait until root is defined
(s5)    **If** the tree is a leaf return
          **otherwise**
                Split the local set according to the
                root values
(s6)    **For** every subset,
          recursively derive the tree
(s7)    Make the decision tree attaching to each
          branch of the root the respective decision
          tree
```

A root will be defined when the message from the Coordinator is received or when the coordinator itself determines the root.

The waiting time could consist a major disadvantage of this approach in step s4

Figure 4 5 illustrates the algorithm



Color Shape Size Class
e1
e2
e3
e4
e5
e6
e7
P1 (Coordinator)
P2
Step 1 & 2 Each processor counts, P2 sends class counts to P1
Processor P1
Processor P2
Coordinator P1
Step 3
Color
Shape
Size
Step 5 Each processor uses the best attribute and splits the data set accordingly
Color
e1 e2 e3
e4
Step 1 & 2 Each processor counts, P2 sends class counts to P1
Coordinator P1
Step 3
Shape
Size
Step 5 Each processor uses the best attribute and splits the data set accordingly
Color
Shape

Figure 4 5 Revised Distributed Tree Derivation

The update phase in a distributed setting is handled as follows. Each processor will receive its respective update data, get its partial class counts and send them over to all other processors or the coordinator. Each processor will receive all class counts and update the tree. In the first approach, each processor will call its reorganization procedure. Computing time will be saved if we use the coordinator approach. In this case, the final tree must be transmitted to all remaining processors.

If one were to use any incremental algorithm, such as the ITI algorithm [45] or the algorithm by Schlimmer [38] and the algorithm is based on class counts, then it is possible to employ the approach proposed here for the derivation and updating every tree incrementally using chunks of updating data (sending class counts for a single case will be more costly than sending the case data).

To get an estimate of the data to be stored in memory (or secondary storage), consider the following parameters: 25 attributes, 100 values per attribute and 100 classes. Then the array of frequencies will be at most 250 000 entries. The expected universal domain for a database with those parameters will be 100^{25} potential entries. Even small subsets will be big enough to make class count interchange beneficial.

It is clear that if a processor keeps only a few tuples, it will be better to transmit those tuples than to transmit the class counts. However, the receiving processor must compute the frequencies and some time can be saved if one uses the idle processors to do this instead of eventually loading the receiving processor with small computations from different sets. It is worth mentioning that other algorithms based on compute and interchange frequencies or class counts to derive association rules in a distributed environment have shown better performance than other approaches [2].

4.5 The Multiple Goal Decision Tree Algorithm

The following algorithm derives the decision tree for a set of attributes in the database It derives the trees breadth first (different from our Revisited Derivation algorithms) and reads the data base once at each level of all trees Thus, we extract all trees with A passes over the database

Multiple Goal Tree Derivation (Initial step)

```
(s2 1 0) **Read**  database and create class counts for each Goal Attribute.
(s2 1 1) **For all**  goal attribute G do
             **If**  all instances have the same value for G,
             the tree is a leaf with value equal to the
             G value, so no further passes are required for G
             **Select**  the best attribute (the root)
             according to a criterion
(s2 1 2) **For each** instance do
          **For each** Goal attribute do
             **Distribute**  the instance according to the
             value of the root attribute
             **Update**  Class Counts for the subtree
(s2 1.3) **Multiple Derive**  the decision subtree for each subset
          of instances
```

Then for each subtree

Multiple Goal Tree Derivation (MGTD algorithm)

```
(s2 1 1) **For all**  goal attribute G do
             **For all**  subset do
             **If**  all instances are of the same Goal value
             the tree is a leaf with value equal to the
             this so no further passes are required
             **Get**  the best attribute (the root)
             according to the criterion
```

```
(s2 1 2) For each instance do
             For each Goal attribute do
             For each subtree Root attribute do
                 Distribute the instance according to the
                 value of the root attribute
                 Update Class Counts for the subtree
(s2 1.3) Multiple Derive the decision subtree for each subset
             of instances.
```

4.6 Non Deterministic Decision Trees

The attribute selection criteria to define roots in the decision tree construction can lead to situations where there are multiple options for a candidate to root. The common approach to solve this situation has been either to choose one option using additional criteria or randomly select any of possible options However for rule extraction (see chapter 6) this option is not adequate because some rules can be ignored by the process. I am proposing the introduction of *non-deterministic trees*. Those are decision trees with several equivalent branches at the same root or subroot but different subtrees The search process is not deterministic since it can branch to several subtrees An instance can lead to several potential leaves or classes Tree construction in this case is not different from the algorithms above Tree testing or updating will proceed on all equivalent branches or subtrees as if there is no difference among them Equivalent subtrees can be discarded when the respective measure differs of the maximum

4.7 Summary

In this chapter I described how part of the problems mentioned in chapter 3 can be solved Extending the algorithms to large data bases requires memory optimization, minimization of I/Os, and the use of incremental approaches Distribution, Parallelism, Multiple

Goals and Non determinism are necessary to process massive amounts of data. The DTD algorithm can successfully obtain the decision tree in every processor for a distributed data base. The DSD algorithm is useful in parallel machines or in environments where file systems are shared among all processors (local area networks, clustered disks). The MGTD algorithm is useful for extracting all data dependencies (rules) simultaneously. The fact that we can use the algorithms both in incremental and non incremental applications makes those approaches very flexible. Using tree reorganization for large data bases looks expensive at first glance, but if the tree has already been derived, non pure incremental methods and tree reorganizations, which are optimistic in nature, seem a fairly good alternative to updating the tree and changing its structure.

CHAPTER 5
THE DETERMINATION MEASURE

5.1 The Determination Criteria

In this chapter, I explain the reasoning behind a new measure for class determination called the determination measure. I discuss its mathematical properties and I show applications of the determination to rank rules and to decision tree construction in large databases.

5.1.1 Fundamentals of the Determination Measure

Classification is the mapping of objects to specific classes. In most applications, this mapping is not unique and an object can be assigned to different classes. Thus, given the relative probabilities of the object for each one of the classes, several measures have been used to evaluate the classification defined for this probabilities [30], [31], [15], [16], [19], [33], [38], [41]. If an object is mapped to a class with high probability and with low probability to other classes, we say it is a good classification, meanwhile mapping with similar probabilities to all classes can not be considered a good one. Among others, the most famous and common measure is information entropy, since the set of n potential classes can be seen as a channel output [30], [38].

$$H_n = -\sum_{i}^{n} p_i log p_i \tag{5.1}$$

where p_i is the relative probability of being in class i

Thus, a low entropy value is interpreted as a minimum amount of uncertainty (high certainty) and a high entropy value as a large uncertainty

However, the entropy used in attribute selection for building decision trees has shown a tendency to select many valued attributes. Besides the entropy value is different when more classes are present and therefore it is difficult or impossible to compare the entropy values for different numbers of classes Take for example the entropy for two classes H_2 and the entropy for three classes H_3 While $0 < H_2 \leq 1$, the entropy H_3 satisfies $0 < H_3 \leq log(3)$ Most of these problems were documented by Quinlan and Arguello [31], [4]

We interested in a measure that, given the probabilities of each class, is able to tell which class is most plausible 1 if there is complete certainty and 0 if not

The information gain criterion or entropy (equation 5 1) can be used to this aim, and its certainty is given by

$$C(\vec{p}) = 1 - \frac{H_n}{log n} \tag{5 2}$$

where n is the number of different classes in the data set

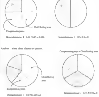
Since the entropy based criterion has several limitations as shown by Quinlan [31] I am proposing an alternative determination criterion given by

$$D(\vec{p}) = 1 - \frac{1}{n-1} \sum_{i \neq j} \frac{p_i}{p_j} \tag{5 3}$$

where $p_j = \max_i p_i$. Equivalently the previous equation can be written

$$D(\vec{p}) = \frac{\frac{1}{n-1}\sum_{i \neq j}(p_j - p_i)}{p_j} \tag{5.4}$$

Intuitively, the determination guesses the most probable class in a given data set based uniquely on their relative probabilities. The presence of elements of other classes precludes the possibility of one class. See figure 5.1 and 5.2



Analysis when two classes are present
Compensating area = Contributing area
+
−
Contributing area
Compensating area
Determination = 1 0.25 / 0.75 = 0.666
Determination = 1 0.5/0.5 = 0
Analysis when three classes are present.
Compensating area = Contributing area
Contributing area
Compensating area
Determination = 1 0.2/0.6 = 0.666
Determination = 1 0.33 / 0.33 = 0

Figure 5.1 The determination measure

This criterion measures the relative importance of the dominant class in a data set (the class with a higher relative probability) with respect to the remaining classes. If the probability of the dominant class is close to those of the remaining classes –differences with it are lower– then the determination is lower. On the contrary, if the probability of the

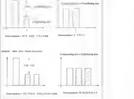

= Contributing area
Determination = (0.75 0.25) 0.75 = 0.666
Determination = 0 0.50 = 0
Analysis when three classes are present
Compensating area = Contributing area
0.80 0.20
Determination = 1/2(2*(0.60 0.20)) / 0.60 = 0.666
Determination = 1/2 (0+0)/0.33 = 0

Figure 5 2 The determination measure

dominant class is in average higher than the remaining classes, then the determination will be higher Thus, a full dominant class leads to a determination of 1 and the absence of a dominant class leads to a zero determination [4].

From a measure point of view, we are measuring the difference of dominant class with respect to each one of the other classes and taking the average over the n − 1 possible values Since the value can be as large as the dominant class value, then we normalized dividing by zero This measure is therefore similar to the square error measure taken over the n − 1 non dominant components and then normalized A potential different measure can be obtained using the square error and normalizing We prefer the simpler and easier to compute one See figure 5 2

The simplicity of the determination formula allows an easy interpretation of its respective values For example although an 80% of certainty based entropy indicates almost nothing about the nature of the dominant class an 80% determination indicates that the dominant class is 5 times higher than the average of the remaining classes Application of the proposed formula shows that if the determination is $1 - \alpha$, then the dominant class, say j satisfies $p_j = \frac{1}{\alpha}\frac{1}{n-1}\sum_{i\neq j} P_i$ $\alpha > 0$

or equivalently

$\delta = D(p_1, p_2, ..., p_n) = \frac{np_{max}-1}{(n-1)p_{max}}$ and therefore $p_{max} = \frac{1}{n-(n-1)\delta}$ Thus when two classes are present a level of 0 80 % confidence can be achieved with a determination of 0 75

Given the distribution of probabilities one can decide which is the best determined class – that with maximum probability , which constitutes the *confidence* in that decision i e , *Confidence rate* $= p_{max}$

Given two sets with the same relative frequencies a way to distinguish between them is to consider their size. Thus, the *support* of a given data set is the data set size The largest set has the maximum support Those concepts will be useful later when dealing with rule extraction in databases

5.1.2 A Mathematical Theory of Determination

This section shows that it is possible to derive mathematically the determination measure on the basis of a limited set of assumptions, using a method similar to Shannon's for deriving his entropy formula [40]

1.3 Assumptions.

Given a set of measures $p_i \geq 0$ [illegible] 1, such that [illegible] of the p_i must be nonzero and there must be at least two possible events if we want to distinguish between them

A measure of determination must satisfy

1 $0 \leq D(p_1, p_2, \ldots, p_n) \leq 1$

2 $D(p_1, p_2, \ldots, p_n) = 0$, if $p_i = p$ for all i

3 $D(p_1, p_2, \ldots, p_n) = 1$, if $p_i > 0$ for some i and $p_j = 0$, for $j \neq i$

4 $D(p - \alpha_1, p - \alpha_2, \ldots, p - \alpha_{n-1}, p) = 1 - D(\alpha_1, \alpha_2, \ldots, \alpha_{n-1}, p)$ $0 \leq \alpha_i \leq p$ $p > 0$

5 $D(p_1, \ldots, p_k, \ldots, p_s, \ldots, p_n) = D(p_1, \ldots, p_s, \ldots, p_k, \ldots, p_n)$

6 $D(C + p_1, C + p_2, \ldots, C + p_n) = D(p_1, p_2, \ldots, p_n)$ $C > 0$

Assumption 1 says that the measure must be be in the range [0 1] meaning that zero is the minimum determination and 1 is the maximum determination

Assumption 2 says that under the same conditions there is no determination

Assumption 3 says that in total discrimination -only one p_i not null the determination must be maximal.

Assumption 4 forces an equal treatment for all measures independant of their coordinates or indexes It allows that under similar conditions the change in determination must be the same Note that the vector $(p - \alpha_1, p - \alpha_2, \ldots, \alpha_{n-1}, p)$ is a distance $d = \sqrt{\sum_i \alpha_i}$ from the vector $(p, p, \ldots, p)$ Similarly the vector $(0, 0, \ldots, 0, p)$ is a distance of d of vector $(\alpha_1, \alpha_2, \ldots, \alpha_{n-1}, p)$ The first change in determination must be equal to the second This corresponds to our intuition that D(0 98,1) and D(0 02,1) are related by D(0 98,1) = 1 D(0 02,1)

Assumption 5 says that the determination function is completely symmetric i e , the interchanges of any two coordinates should not affect the result

Assumption 6 says that multiplying the measures for any constant should not affect the result since the relative composition of the measures is not affected

Note that the particular case, when $\sum p_i = 1$ represents a distribution of probabilities and therefore the determination can be applied in the same way Sometimes it is useful to think in normalized determination i.e , when its arguments can be seen as a set of probabilities

One objective is to introduce a function that satisfies assumption 1 through 6 and is simple to compute i e as polynomial or fractional representation

5.1.1 Derivation of the Determination Function

Theorem 1

If n=2 , a polynomial measure satisfying assumptions 1 through 6 does not exist

Proof

Assume that the Determination formula is of the following form $D(x_1, x_2) = \sum_i a_i x_1^{r_i} + \sum_i b_i x_2^{s_i} + \sum_i c_i x_1^{t_{1i}} x_2^{t_{2i}} + B$ (1 1) with all exponents positives integers and non zero exponents in the third term

Using condition 3 and 6 $D(k, 0) = \sum_i a_i * k^{r_i} + B = 1$ $D(0, k) = \sum_i b_i * k^{s_i} + B = 1$

This must be true for every $k > 0$ Two polynomials are equal if all their coefficients are equal and thus all a_i and b_i are zero and B=1

Then, by condition 2 $D(C, C) = 1 + \sum_i$ [illegible] $= 0$ and again for condition 6 this is valid for all C therefore, there should exist a subset of indexes such that $t_{1,i} = -t_{2,i}$. Since the exponents are all positive, such a condition is not possible, and hence there is no such polynomial

Theorem 2.

If $n=2$, a measure that satisfy conditions 1 through 6 is

$D(x_1, x_2) = \max(1 - x_1/x_2, 1 - x_2/x_1)$

(taking division by zero as a limit to positive infinity)

Proof

Without loss of generality, let us assume that $0 \leq x_1 \leq x_2$. Thus, $D(x_1, x_2) = 1 - \frac{x_1}{x_2}$ since $1 - x_2/x_1 \leq 0$ (or its limit when x_1 tends to zero)

Thus, assumption 1 holds $0 \leq 1 - \frac{x_1}{x_2} \leq 1$

Assumption 2 if $x_1 = x_2$, then $D(x_1, x_2) = 0$

Assumption 3 $D(0, x_2) = 1 - 0/x_2 = 1$ for every $x_2 > 0$

Assumption 4 $D(x_2 - a, x_2) = 1 - \frac{x_2 - a}{x_2} = 1 - (1 - \frac{a}{x_2}) = 1 - D(a, x_2)$

Assumption 5. The interchange of variables doesn't change the sign of the inequality $x_1 \leq x_2$, and then assumption 5 holds.

Assumption 6 $D(C * x_1, C * x_2) = 1 - \frac{C x_1}{C x_2} = 1 - \frac{x_1}{x_2} = D(x_1, x_2)$ //

The previous theorem does not guarantee the uniqueness of the function, but simply says that the given formula is adequate

Theorem 3.

For $n > 0$, no polynomial measure satisfies conditions 1 through 6

Proof

A possible polynomial function can be expressed in the following form

$$D(\vec{p}) = \sum_j (\sum_i \alpha_{i,j} p_j^{r_{i,j}}) + \sum_k \beta_k (\prod_l p_l^{s_{l,k}}) + K \tag{5.5}$$

with

$$r_{i,j} \neq 0 \ \forall i, j \tag{5.6}$$

and there are at least two $s_{l,k} > 0$ for a given k

Using conditions 3 y 6

$D(C * \vec{e_j}) = \sum_i \alpha_{i,j} C^{r_{i,j}} + k = 1$

This must be true for every $C > 0$, then for 5.6 all

$\alpha_{i,j} = 0$ and $K = 1$

Thus 5.5 becomes

$$D(\vec{p}) = \sum_k \beta_k (\prod_l p_l^{s_{l,k}}) + 1 \tag{5.7}$$

Then, by condition 2

$D(C, C, \ldots, C) = 1 + \sum_k \beta_k C^{\sum_l s_{l,k}} = 0$

and again for condition 6, this is valid for all constant vectors C , there should exist a subset of k, S, such that those three conditions hold

$$\sum_{k \in S} \beta_k = -1 \tag{5.8}$$

$$\beta_k = 0, k \text{ not } \in S \tag{5.9}$$

$$\sum a_{i,k} = 0 \forall k \in S \tag{5 10}$$

Since there exist at least two $a_{i,k} > 0$ for each k, 5 10 can not hold and such a polynomial therefore does not exist //

Theorem 4

Given $n > 0$, a measure that satisfy conditions 1 through 6 is $D(\vec{p}) = 1 - \frac{1}{n-1} * \sum_{i \neq z} \frac{p_i}{p_z}$ where $p_z = \max_i p_i$

Proof I limit our analysis to the subspace where p_n is maximum

Assumption 1

$\sum_{i \neq n} p_i \leq (n-1) * p_n$ since p_n is maximum

Then

$\frac{1}{n-1} \sum_{i \neq n} \frac{p_i}{p_n} \leq 1$ since $p_n > 0$

and thus $D(\vec{p}) \geq 0$

$-\frac{1}{n-1} \sum_{i \neq n} \frac{p_i}{p_n} \leq 0$ since all p_i are positive which implies

$D(\vec{p}) \leq 1$

Assumption 2:

$\sum_{i \neq n} p_i = (n-1) * P$ since $p_i = P$ for all i

$D(\vec{p}) = 1 - \frac{1}{(n-1)}(n-1) * P/P = 0$

Assumption 3

$D(0\ 0 \ldots 0, p_n) = 1 - \frac{1}{(n-1)} * 0/p_n = 1$

Assumption 4

$D(P - \alpha_1\ P - \alpha_2 \ldots P - \alpha_{n-1}, P) = 1 - \frac{1}{(n-1)} \sum_{i \neq n} \frac{(P - \alpha_i)}{P} =$

$1 - \frac{1}{(n-1)} \sum_{i \neq n} P - \frac{1}{(n-1)} \sum_{i \neq n} \frac{\alpha_i}{P} = 1 - (1 - \frac{1}{(n-1)} \sum_{i \neq n} \frac{\alpha_i}{P}) =$

$1 - D(a_1, a_2, \ldots, a_{n-1}, P)$

Assumption 5

The interchange of variables doesn't change the sign of the inequality $x_i \leq x_n$, and then assumption 5 holds

Assumption 6

$D(C * \vec{p}) = 1 - \frac{1}{(n-1)} \sum_{i \neq n} \frac{Cp_i}{Cp_n}$ for $C > 0$ Cp_n is still maximum

Therefore $D(C * \vec{p}) = D(\vec{p})$

5.2 Application of the Determination Measure for Rule Evaluation

In this section I show how the determination can be use to rank rules for rule induction. The general problem is defined by the following

For a set of probabilistic rules of the form if Y = y then X = x with probability p_i one is interested in determining which rule is most appropriate.

Smyth and Goodman used cross-entropy to evaluate rules. Here is a comparison of the determination measure and its use for ranking rules with the measure supplied by Smyth and Goodman

The value of cross-entropy is defined as

$j(X, Y = y) = p(x/y)log(\frac{p(x/y)}{p(x)}) + (1 - p(x/y))log(\frac{1-p(x/y)}{1-p(x)})$ and the J measure $J(X, Y = y) = p(y)j(X, Y = y)$ [41]

The determination will be

$det(X, Y = y) = max(1 - \frac{P(x/y)}{P(\neg x/y)}, 1 - \frac{P(\neg x/y)}{P(x/y)})$ and $Det(X, Y = y) = p(y)det(X, Y = y)$

The following example is due to Smyth and Goodman [41, pp 164-165] (I have added determination measures to the tables.)

Table 5 1 Joint probability distribution for Medical Diagnosis example

Symptom A	Symptom B	Disease X	Joint Prob
no fever	no sore throat	absent	0.20
no fever	no sore throat	present	0 00
no fever	sore throat	absent	0 30
no fever	sore throat	present	0 10
fever	no sore throat	absent	0 02
fever	no sore throat	present	0 08
fever	sore throat	absent	0 03
fever	sore throat	present	0 27

Table 5 1 shows the probability distribution of medical cases for diagnosis of a Disease X Table 5 2 shows a set of potential rules and the evaluation of each rule using both the J measure and the determinateness measures shown above The similarity between both results must be noted However the required computation of the determinateness measure is much less than the computation of the J measure So, this is significant when innumerable amount of extract rules needs to be evaluated to discriminate among them This is the case when a decision tree is being constructed from a large database

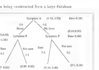


Figure 5.3 A decision tree and corresponding rules

Table 5.2 Rules and their information Content (Determination measures added)

Num	Rule	$p(x\|y)$	$p(y)$	$j(X;y)$	$J(X;y)$	$det(X;y)$	$Det(X;y)$
1	if fever then disease x	0.875	0.4	0.672	0.238	0.86	0.344
2	if sore throat then disease x	0.528	0.7	0.018	0.012	0.108	0.075
3	if sore throat and fever then disease x	0.8	0.3	0.654	0.196	0.868	0.260
4	if sore throat and no fever then not disease x	0.75	0.4	0.124	0.048	0.656	0.256
5	if no sore throat and no fever then not disease x	1.0	0.2	0.863	0.173	1.0	0.2
6	if sore throat or fever then disease x	0.5825	0.8	0.037	0.030	0.222	0.177

Both criteria choose the first rule as most conclusive. The third rule is just a subcase of the first one. There is a difference with the fourth rule which seems due to the parameter symmetry.

It is worth noting that the previous set of rules (except rules 2 and 6) can be seen as a decision tree (shown in Figure 5.3) in which the nodes identify the conditions in the precedent and the leaves represent the final outcome.

Each branch can be evaluated using the respective criterion and the most conclusive rule extracted. The number in each node denote the numbers shown in the previous tables.

5.3 Application to Classification in Large Databases

5.3.1 Influence of Many Valued and Irrelevant Attributes

Despite many studies that show the tendency of the entropy to favor many valued attributes [31] [27] I am including several experiments to analyze the effect of many valued attributes in entropy and determination for large databases. The experiments are divided in two parts. The first two experiments use the determination criteria and entropy criteria. The other three experiments use a modified version of the entropy [31] called the gain ratio use (similarly a modified ver[illegible] to favor few valued attributes It will be explained later

The data

The synthetic databases for the experiment consisted of 20000 cases each 20 attributes There were 4 databases the two first databases for the first part of the experiment The last two database, together with the first one are used for the second part of the experiment

1 The data generation program was instructed to generated 10 class values and to leave 5 attributes as irrelevant for class assignment i.e., those attributes are not used in computing the distance of the tuple to the respective class centroid Those attributes (A15 to A19) have a range of 30 for attribute A15 and 250 for attributes A16 to A19 The remaining attributes were relevant to the class value and all of these have 2 or 3 values The data were generated using a modified version of the DGP/v2 of P Benedict [7] Despite the random nature of the program there is no guarantee that a random dependency of the Class attribute with the irrelevant variables can not be introduced

2 In this database 10 attributes were left as irrelevant and to facilitate the induction task only two classes were used The irrelevant attributes had a cardinality around 500 values While in the previous database, the five many valued attributes had little chance to be chosen, in this database the 10 many valued attributes had a major chance.

3 Again, 10 classes were generated and this time there were 10 irrelevant attributes From the 10 relevant attributes five were chosen as many valued (A5 to A9 with 250

relevant. From the 10 irrelevant attributes five were chosen as many valued (A15 to A19 with 250 values).

Contrary to the previous databases, this time the relevant attributes were chosen as *many valued*. The second experiment tries to show how the *gain ratio* or *few-valued determination* are biased to few valued attributes even when those are irrelevant.

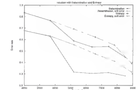


Figure 5.4 Many values Experiment 1

The first experiment

The first part of the experiment was designed to detect the influence of many valued attributes in both criteria entropy and determination. The experiment consisted of several iterations, starting with a sample size of 10% (2000) of the cases. The error rate and soft error rate (sometimes called the error caused by undefined cases i.e. error caused for cases out of domain of the decision tree) for both entropy and determination is depicted in figure 5.4. It can be noted how the error rate for entropy is higher. Convergence i.e. a relatively

Table 5.3 Tree Characteristics for many values experiment 1

Determination							
Iter	Root	Measure	Closest	Measure	Height	Nodes	Leaves
1	A18	0 760	A19	0.745	3	2125	1918
2	A18	0 706	A16	0.705	3	3913	3565
3	A15	0 697	A16	0.684	5	1901	4074
4	A15	0 685	A18	0.672	5	5588	4750
5	A15	0 681	A19	0.650	4	5763	4847
6	A18	0 679	A18	0 668	5	5934	6130
Entropy							
Iter	Root	Measure	Closest	Measure	Height	Nodes	Leaves
1	A18	0 343	A19	0.304	3	2125	1918
2	A18	0 286	A19	0.237	3	3920	3572
3	A18	0 243	A16	0.213	3	5521	4818
4	A18	0 228	A15	0 213	3	5315	5148
5	A19	0.220	A15	0.213	3	5476	5318
6	A1	0.218	A18	0 214	8	3299	3075

stable low error rate – is obtained when almost 50% or more of the cases are included in the sample for entropy while determination tends to get a lower error rate after the second iteration

Table 5 3.1 shows the tree characteristics At the beginning both criteria tend to favor A18 After that, it must be noted that while the determination sticks to the same attribute (A15 with 30 values), entropy favors A18 with 250 values At the end, entropy changes the root selected to a relevant attribute due to the high relative size of the sample A1 corresponds to root of the final tree for the complete 20000 cases for entropy, while A15 corresponds to the root of the final tree for determination It is interesting to note that even though A15 was marked as irrelevant, the final fact is that there is an association between A15 and the class (as the decision tree says) I believe this was primarily due to the few values of A15 (30) and to a coincidence of the normal distribution used for the data generation program Note that A18 is still important as a closest attribute for the root in Table 5 3 1 for entropy

In a second trial, the second synthetic database was used Figure 5 5 shows the results for 12 or 13 iterations However the error rate is lower in both cases, due to the lower number of classes (2) the determination error rate is generally the lowest Table 5 3 1

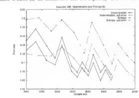


Figure 5 5 Many values Experiment 2

shows the roots for the six first iterations While both criteria choose as roots a few valued attribute (A0 to A9) it seems the lowest height of the entropy trees and the large number of nodes that some many valued attributes were chosen as subroots in the subtrees and hence the large error rate Note that the soft error rate is generally lower for determination

The second experiment

In order to avoid the negative effect of the many valued attributes for entropy Quinlan suggested the gain ratio criteria [31] Thus, for the selection step of the tree construction

Table 5 4 Tree Characteristics for many values experiment 2

Determination				
Iter	Root	Height	Nodes	Leaves
1	A2	6	413	390
2	A2	7	876	824
3	A7	7	1139	1062
4	A2	8	892	843
5	A2	8	1375	1314
6	A2	8	1050	1110
Entropy				
Iter	Root	Height	Nodes	Leaves
1	A0	3	812	715
2	A6	3	1109	947
3	A0	4	1283	1114
4	A0	3	1464	1277
5	A6	4	1671	1417
6	A0	6	1984	1706

algorithm the best attribute is selected as the attribute A that maximizes

$$\frac{(H_n(S) - EH_n(A))}{IV(A)} \tag{5 11}$$

where

$H_n(S)$ is the entropy according to the class distribution of the data set of instances S

$EH_n(A) = \sum P(A = a_i) * H_n(A = a_i)$ (average class entropy for the attribute A)

$IV(A)$ is the randomness measure of the partition caused by A i.e , the entropy of the subsets $A = a_i$ over S

Note that this equation will favor low valued attributes, since the largest value of $IV(A)$ will be $\log(|A|)s$, and the information gain (numerator) will be reduced in those cases

A similar component is introduced here for the determination formula, the root attribute will maximize

$$D_f(\vec{p}) = \frac{D(\vec{p})}{|A|/MC} = MC * \frac{D(\vec{p})}{|A|} \tag{5 12}$$

where $MC = \max_A |A|$

Note that this equation reduces the determination of an attribute according to its relative number of values. In an environment where all attributes have the same number of values the formula coincides with the basic formula

Both criteria were used for this second part of the experiment

The first trial

Using the first database several iterations were made for both criteria Figure 5 6 shows the error rate for both few valued determination and the gain ratio. In this case the gain

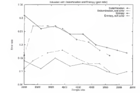


Figure 5 6 Many values Experiment 3

ratio tends to outperform the determination modified but the soft error rate is still lower for determination. Note that the reduction in the error rate from the figure 5.4

Table 5 3 Tree Characteristics for many values experiment 3

Determination				
Iter	Root	Height	Nodes	Leaves
1	A2	12	922	661
2	A12	12	1232	888
3	A12	11	1609	1201
4	A12	14	1862	1417
5	A12	13	2229	1653
6	A12	14	2578	1963
Entropy				
Iter	Root	Height	Nodes	Leaves
1	A12	11	3306	2648
2	A12	11	987	738
3	A1	18	1452	1163
4	A12	11	1800	1451
5	A1	11	2260	1822
6	A12	11	2538	2052

Table 5 3 1 shows the roots selected in the first six iterations The criteria effectively choose few-values attributes rather than many valued attribute as expected Trees are more compact in terms of the number of nodes than in the previous run in the first part of the experiment

The second trial

Using the third synthetic database several iterations were made for both criteria Figure 5 7 shows the error rate for both few-valued determination and the gain ratio In this case, the gain ratio outperforms the determination modified, but the soft error rate is still lower for determination The behavior of the gain ratio is consistent but determination behaves randomly

Table 5 3 1 shows the roots selected in the first six iterations The gain ratio criterion effectively chooses few values attributes rather than many valued attribute as expected

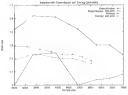


Figure 5 7 Many values Experiment 4

Note that A13 is a few valued irrelevant attribute (thought it helps in the final classification as shown for the error rate of the tree) With the few valued determination, the bias was not so obvious. A later inspection of the decision trees showed that few valued attributes were chosen as subtrees in the determination experiment and hence the large error rate None of the criteria selected many valued irrelevant attributes

The third trial

Using the third database several iterations were made for both criteria. Figure 5.8 shows the error rate for both few valued determination and the gain ratio In this case, the gain ratio tends to outperform the determination modified, but the soft error rate is still lower for determination.

Note the tendency of the both errors of the entropy to be "inside" between both errors of determination for the Figure 5.6 5 7 and 5 8

Table 5.6 Tree Characteristics for many values experiment 4

Determination				
Iter	Root	Height	Nodes	Leaves
1	none	0	1	1
2	A5	5	1747	1118
3	A6	7	2894	1919
4	none	9	1	0
5	A0	10	434	343
6	A0	10	698	548
Entropy				
Iter	Root	Height	Nodes	Leaves
1	A14	10	1003	2648
2	A13	11	1552	738
3	A0	11	1884	1163
4	A0	11	2377	1451
5	A0	11	2645	1822
6	A13	11	3648	2052

Table 5.7 Tree Characteristics for many-values experiment 5

Determination				
Iter	Root	Height	Nodes	Leaves
1	A14	12	922	661
2	A12	12	1232	888
3	A19	11	1609	1201
4	A19	14	1863	1417
5	A19	13	2229	1653
6	A19	14	2578	1863
Entropy				
Iter	Root	Height	Nodes	Leaves
1	A12	11	3398	2648
2	A12	11	987	738
3	A12	10	1452	1163
4	A12	11	1806	1451
5	A12	11	2260	1822
6	A12	11	2538	2052

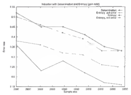


Figure 5 8 Many values Experiment 5

Table 5 3 1 shows the roots selected in the first six iterations The criteria effectively choose few values attributes rather than many valued attribute as expected even though those were irrelevant An inspection of the generated trees, shows that most of the nodes consisted fo few valued irrelevant attributes rather than the relevant attributes (hence the large error rate)

In conclusion when there are irrelevant many-valued attributes present

1 Error rates are high due to those many valued attributes

2. Entropy tends to be erratic with small samples relative to cardinality of the many valued attribute domain

3 Although determination tends to have a lower error rate than entropy the soft error rate is still high because of the influence of many valued attributes However, this is small compared with the entropy and the gain ratio

4 The misclassification error is given for the difference between the two curves in each criterion Entropy seems to have a low misclassification error but a large soft error (undefined error) in all cases

5 Many-valued attributes tend to be chosen as roots mainly in subtrees when the size of the local sets is smaller

6 Although the experiment was conducted with a relatively small set of 20000 cases the results show how the induction for a very large data base can be affected by many-valued attributes Eventually any large data base will be partitioned in small subsets and the induction on these will be affected for many valued attributes. Even though, the error rate in large databases will be only lightly affected because relevant attributes will be chosen in higher levels of the decision tree

5.4 Comparing Entropy and Determination

The next experiment was designed with the aim to compare the inductive ability of simple determination with the entropy in an environment where the entropy (and internal nature) are not affected by many valued attributes or irrelevant ones.

5.4.1 Generation of Experimental Databases

Four synthetic databases with around one hundred thousand cases (tuples) were generated for the experiments Each database consisted of 20 attributes (A0 to A19), the class attribute and 10 values per attribute (0 to 9) approximately avoiding the effects of many valued attributes described in the previous section The way the values of the class are assigned determines the type of the database as described below

The first database includes just two classes. The main class consists of all those instances around a random peak in the 29 dimensional space generated by the data generation program developed by Powell Benedict and others [7].

The second database was developed using the same program but modified to generate 10 class groups, to substantially complicate the task of the decision tree induction algorithm.

The classes in the third database were generated at random. Actually, one attribute (the first) was chosen as the class designator since its values were generated at random.

For the last database classes were designated using a decision tree known beforehand. An initial database was generated and class values were changed accordingly to the decision tree output. This case represents a database that has a well-defined and known decision tree embedded in it.

5.4.2 Experiments

Four experiments were conducted to demonstrate

- That the proposed determination criterion compares well with the entropy based criterion
- The applicability of decision tree approach to large databases (as they have been previously used mostly for small learning sets) and
- The effectiveness of the use of a small sample set (instead of the entire database) for knowledge discovery

Each experiment was performed with a synthetic database described above. Each experiment consisted of a set of tree inductions. Each tree induction was determined for the initial sample tuple set (percentage of which shown as part of the table and varies from 2 to 17 %) taken from the database Once the actual decision tree is derived for the sample set the rest of the database is tested against the tree and the error rate computed Then a percentage of the exceptions is used to reconstruct the decision tree and again the rest of the database was tested This inductive process continued up to a predefined number of iterations (one in some cases) The process was halted either if the error rate was low enough or if only a slight improvement over the previous error rate was computed for the current tree A high error rate could be the result of a partition which cannot be best described in terms of the induced decision tree On the other hand, reasonable improvements of error rate with windows in which exceptions are *included* indicate convergence towards the appropriate decision tree for the entire data The above experiment is designed to understand the effectiveness of the initial sample and the rate of decrease of error when exceptions are added to the initial sample

5.4.3 Updating the Window through Sampling of Exceptions

The original algorithm requires that all exceptions in the current window be incorporated into each iteration (step #2 3) When dealing with large databases, it is more realistic to incorporate a small percentage of the exceptions in each iteration A small sample that exhibits uniform distribution of the exceptions seems the best option In the implementation a parameter to the induction process is provided to select this small sample of the exceptional cases The reason behind this is to keep the window size small since many exceptions can be due to the same cause (a wrongly labeled *leaf* in a missing branch) This can lead to a slow convergence in some cases but it avoids superfluous data in the window

Terminology

Set The initial sample set with which the induction process starts All sample sets are taken uniformly distributed over the synthetic database This guarantees a meaningful sample from the database The table indicates those cases where a different initial sampling method was used

Sample size The number of cases in the initial sample

Initial error The initial classification error when the rest of the database was tested against the tree derived for the initial sample

N It Number of Iterations done to get a final tree (by including the exceptions added for each iteration)

% Ex Sd % of Exceptions Sampled Percentage of exceptions that are added to the window after each iteration

Final error The final tree error measured with the rest of the database

Final S Size Final sample size that includes all the exceptions that were added in each iteration

Tree Size Tree Ht (Height) Tree Leaves Tree Nodes the decision tree features. Size is given in kilo-bytes and mega-bytes for the tree in memory

Root and Root Meas The decision tree root attribute and the measure either determination or certainty based entropy

Ext Tr The number of external trees required when the tree no longer fits in memory Only the main body of the tree is left in memory

Other terminology used but not shown in the table

Soft cases tree exceptions due to missing tree branches These constitute a main source of exceptions when dealing with attributes that have large domains

5.4.4 Test Results

Experiment #1 Database 100000 records two classes 20 attributes, 10 values per attribute. See tables 5.8 and 5.9

Table 5.8 Exp 1 Criterion Determination

Set	Sample size	Initial error	N St	% Er Sd	Final error	Final E-time	Tree time	Tree Ht	Tree Leaves	Tree Nodes	Tree Root	Root Mean	Ext Tr
0 (1)	200	7 26	3		0 33	1582	97k	6	853	1059	A11	0 98	9
1	2000	1 20	10	3 1	0 11	4011	1 0m	1	1756	3403	A11	0 01	0
1	5000	0 107	1	3 0	1 184	1108	1 0m	1	3831	1103	A11	0 0%	0
1 (2)	10000	0 140	1	1	1 117	14848	4 3m	1	4063	5009	A18	0 03	15

Table 5.9 Exp 1 Criterion Entropy

Sr	Sample size	Initial error	N St	% Er Sd	Final e rr	Final E-time	Tree Size	Tree Ht	Tree Leaves	Tree Nodes	Tree Root	Root Mean	Ext Tr
1 (1)	158	1 48	1		0 01	2161	855k	6	735	968	A3 (*)	7 10	0
1	2000	0 188	10	2.5	1 161	6935	1 1m	1	3114	1534	A55	1 34	1
1	5000	1 10	1	2.1	1 111	1271	1 1m	1	1120	3808	11	1 207	1
1 (1)	10000	1 185	1	1	1 121	7130	11 m	1	3461	1323	11	1 30	10

In all cases soft errors make up 40% of the final error

(1) The initial error has approximately 10% soft cases An intermediate sample of 5000 rows was used to test the trees First 1145 (1357 for entropy) rows were added, and then 787 (445 for entropy) rows were added The approach was abandoned since 1 was approximating the tree from the 5000 rows sample which had a 0 10 and 0 15 percent error respectively (See Set 2)

(2) 13 subtrees having an average 4 nodes, 4 leaves and 1 level were stored in external files (secondary storage) The average tree size was 4k bytes

(3) 19 subtrees having an average of 4 nodes 3 leaves and 1 level were stored in external files The average tree size was 3k bytes

(*) A posterior check of the decision tree shows that A16 has the same root measure as that of A3 (A3 was chosen by lexicographical order)

Observations

- Final errors can be reduced by almost 40% if the missing branches corresponding to the soft cases are added to the final tree

To obtain 11% (say from 25% to 14%) improvement in error it is necessary to increase the sample size by almost 7 times (from 2000 to 14000)

Both criteria lead to an error rate of 15% or better

Although the entropy behaves a better than determination (2% better), the decision tree is different from set to set (see Root column in the entropy case) indicating a random behavior while there is more stability of the tree for determination. The final trees have different root and attributes although they tend to be similar in size, height, number of nodes and leaves

Experiment #2: Database 100000 records, 10 classes 20 attributes 20 values per attribute See tables 5 10 and 5 11

Table 5 10 Exp 2 Criterion Determination

Set	Sample size	Initial error	E D	% Ex Sd	Final error	Final S size	Tree Size	Tree H	Tree Leaves	Tree Nodes	Tree Root	Root Meas	Ext Tr
4	16000	0.35	3	5	0.21	14095	6.7m	7	5876	7286	A5	0.81	536 (3)
5	17000	0.28	1		0.24	17000	6.7m	7	5258	7858	A3	0.78	473

Table 5 11 Exp 2 Criterion Entropy

Set	Sample Size	Initial erro	E D	% Ex Sd	Final erro	Final S Size	Tree Size	Tree Ht	Tree Leaves	Tree Nodes	Tree Root	Root Meas	Ext Tr
4	16000	0.31	3	5	0.26	15879	6.7m	5	5491	7835	A13	1.388	506 (3)
5	17000	0.26	1		0.28	17000	6.2m	7	6888	7758	A13	0.590	353

In all cases soft errors make up 50% of the final error

Table 5.12. Exp 3 Criterion Determination

Set	Sample size	Initial error	# It	% Ex Sd	Final error	Final S Size	Tree Size	Tree Ht	Tree Leaves	Tree Nodes	Tree Root	Root Mem	Est Tr
6	[illegible]	0.71	1	6	0.71	10000	6.2m	11	7377	9986	A5	0.469	620 (1)
7	7000	0.66	1	6	0.66	17000	6.6m	16	8587	11123	A5	0.469	1958 (2)

Table 5.13 Exp 3 Criterion Entropy

Sd	Sample size	Initial error	# It	% Ex Sd	Final error	Final S Size	Tree Size	Tree Ht	Tree Leaves	Tree Nodes	Tree Root	Root Mem	Est Tr
6	10000	0.70	1	6	0.75	10000	4.3m	20	7313	9572	A5	0.431	368 (1)
7	17000	0.64	1	6	0.66	17000	6.6m	20	8665	11220	A5	0.436	1910 (2)

(1) Subtrees having an average 3 or 4 nodes 3 leaves and 1 level were stored in external files The average tree size was 3k bytes

(2) Subtrees having an average 4 or 5 nodes 3 leaves, and 1 level were stored in external files The average tree size was 4k bytes with slight variations

Observations:

Again, certainty based entropy and determination tends to lead to different decision trees but with similar structures (nodes, leaves height and size)

The accuracy tends to be a little better (2%) for certainty based entropy than for determination but both the trees are of similar accuracy (28% to 30%)

Larger samples were not analysed since they require many iterations for a significant accuracy improvement

Experiment #3: Database 100000 records 10 classes 20 attributes, 18 values per attribute A random class definition See tables 5 12 and 5 13

In all cases soft errors make up 40% of the final error

(1) Subtrees having an average 3 nodes 3 leaves and 1 level were stored in external files The average tree size was 3k bytes

Table 5.14 Exp 4 Criterion Determination

Set	Sample size	Initial error	# It	% Ex Sd	Final error	Final # Ex	Tree Size	Tree Ht	Tree Leaves	Tree Nodes	Tree Root	Root Mean	Ext Tr
6	1726	0.000	1	4	0.055	1726	31k	3	29	32	A1	0.999	6
9	172	1.11		0	1.403	172	14k	3	25	28	A1	1.94	1
1 (4)	172	0.35	3	2	0.081	1100	11k	3	30	33	A1	1.38	1

Table 5.15 Exp 4 Criterion Entropy

Set	Sample size	Initial error	# It	% Ex Sd	Final error	Final # Ex	Tree Size	Tree Ht	Tree Leaves	Tree Nodes	Tree Root	Root Mean	Ext Tr
6	1726	1.000	1	4	0.055	1726	31k	3	29	32	A1	0.835	6
9	172	1.11		0	1.403	172	14k	3	25	28	A1	0.15	1
1 (4)	172	1.11	1	1	0.000	550	20k	3	30	33	A1	0.86	1

(2) Subtrees having an average 5 nodes (several trees with 7 or 20 nodes), 3 leaves and 1 level were stored in external files The average tree size was 4k bytes.

Observations

Both criteria behave similarly Their ability to correctly classify the cases are equally bad (due to the random class assignment)

Final decision trees were similar in both cases

Experiment #4 Database 86418 records, 20 attributes, embedded decision tree class The embedded decision tree had the following characteristics: size 84k height 2, leaves 39 nodes 42 root A1 See tables 5 14 and 5 15

In all cases soft errors make up 100% of the final error

(1) This set constitutes the first 172 cases of the artificial database

Observations

Definitely the embedded decision tree was detected easily for both criteria. A very small sample of 172 (0 2%) leads to an almost exact decision tree (0 018% error)

Even a bad sample (set 10) leads to an exact decision tree after 2 or 3 iterations in both cases

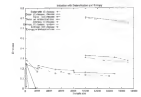


Figure 5.9 Experiment results

5.4.5 Summary

Figure 5.9 shows the relationship between sample size and error rate for both determination and entropy Each line represents an experiment and shows the convergence of the induction process Determination shows a very close behavior to entropy (an average 2% difference) These synthetic databases represent a good context for induction with entropy i e there are not many valued attributes present which interfere with the induction process

From the experiments it is clear that decision tree induction in large databases looks like a good alternative to extract rules without the expense of processing the whole database However this can be done with any attribute selection criteria, it is important the computing time invested the behavior of the attribute selection criteria in the presence of many–valued attributes and its relationship with confidence and support Determination is fast to compute than entropy and can be easily adapted in the presence of many–valued attributes ($D(\vec{p})$ and $D_f(\vec{p})$ coincide if all attributes has the same cardinality) Besides the undefined error (soft error) when many-valued attributes were considered was smaller than the entropy case This suggests that the determination will fit better the domain of cases however its ability to classify correctly cases inside the decision tree domain is sometimes much lower than entropy (classification error) leading to a larger error rate in some experiments The fact we can relate the node determination measure with the confidence and support allows for an easy interpretation of observed results (or classification) as compared to the other criterion

On the other hand, if a simple decision tree exists a small sample should be able to detect it with great accuracy If there is not such decision tree, any small sample will lead to an inexact (50% error or more) decision tree

The experiments were carried out on a Sun Workstation with 8 megabyte of memory in a multi user environment Execution times were around 20 to 35 minutes for deriving the decision tree for larger sets (20000 cases) and similar times to test the respective decision tree against the whole database depending on load Similar results were obtained in a Pentium based PC computer

The similar performance of both criteria in the host environment for entropy and the host performance of determination when many valued relevant attributes are present suggest that determination is a viable alternate criteria The biased criteria (gain ratio and few value determination) can selected few valued attributes that are not important or relevant for classification and hence for the derived rules The ability of the determination to have a low soft or undefined cases error is good if we like cover the large amount of cases with the classification outcome from the tree

CHAPTER 6
DECISION TREES AND ASSOCIATION RULES

6.1 Decision Trees, Functional Dependencies and Association Rules

Knowledge Discovery consists mainly of finding rules among data Here I formalize the concept of association rules, their relationship to functional dependencies and decision trees

6.1.1 Confidence and Support in Decision Trees

Definition 1 *A path* P *in a decision tree is a sequence of attribute value pairs denoted* $\{A = a, B = b, \ldots, R = r\}$

A path is simple if it consists of just one attribute value pair

Definition 2 *A leaf is determined by the path to it* P *It is denoted* $L(P)$

Definition 3 *The confidence on the decision represented by a leaf* $L(P)$ *is denoted by* $\mathcal{C}(L(P))$ *and corresponds to the dominant class (the class with large number of elements) in the set denoted by* $\mathcal{D}(L(P))$

Definition 4 *The support of the decision represented by the leaf* $L(P)$ *is given by its cardinality* $|L(P)|$

6.1.2 Definition of Association Rules

Date describes a *functional dependency* among attributes or features in a database as an attribute B depends functionally on an attribute A, if, for every value of A, every tuple that contains this value of A, always contains the same value for B [9]

Mathematically, if $\mathcal{D}(X)$ denotes the *domain* of an attribute X $\mathcal{U}$ denotes the data base and $\tau.B$ denotes the value of attribute B in tuple τ

$\forall a \in \mathcal{D}(\mathcal{A}), \forall \tau, \rho \in \mathcal{U}$ such that $a \in \tau, a \in \rho \Rightarrow \tau.B = \rho.B$

This functional dependency (fd) is denoted as $A \mapsto B$

It is interesting to analyse the meaning of a functional dependency from the point of view of Knowledge Discovery

First the data base $\mathcal{U}$ is generally dynamic We don't know all tuples in a given instant So, we may say that the a $A \mapsto$ B is true for a large known set of tuples Thus, the mathematical concept is no longer applicable (or we need a relaxed notion of functional dependency) but we are still interested in these kind of relationships

Second, even so, the dependency of B in A may not hold for all values of A but for most of them This is not a problem since we can consider a more restricted domain for A However there can still be values of A in which the dependency is true for most of the tuples containing those values i.e., a large subset of the known tuples and we wouldn't like to discard those values Again, the mathematical definition does not hold, but the relationships are still interesting

Let the "large known set of tuples" $\mathcal{S}$ be the *support set*, and "the large subset of the known tuples" $\mathcal{C}$ the *confidence set* Thus, the concept of an *association rule* can be defined as

Table 6 1 Medical Diagnosis example

Num	*Symptom A*	*Symptom B*	*Disease X*
1	no fever	no sore throat	absent
2	no fever	no sore throat	absent
3	no fever	sore throat	absent
4	no fever	sore throat	absent
5	no fever	sore throat	absent
6	no fever	sore throat	present
7	fever	no sore throat	absent
8	fever	sore throat	present
9	fever	sore throat	present
10	fever	sore throat	present

be $\mathcal{N}$ the set of natural numbers Given values $c \in [0\ 1]$ and $s \in \mathcal{N}$,

If $\exists \mathcal{S} \subset \mathcal{U} / \mid \mathcal{S} \mid \geq s$ and $\exists \mathcal{C} \subset \mathcal{S} / \mid \mathcal{C} \mid \geq c * \mid \mathcal{S} \mid$

such that

if $\mathcal{R} = \{\S \in \mathcal{D}(\mathcal{A}) / \exists \tau \in \mathcal{S}, \S \in \tau\}$

(the set of values of A restricted to the set of tuples S) then

$\forall a \in \mathcal{R}\ \forall \tau, \rho \in \mathcal{C} / a \in \tau, a \in \rho \Rightarrow \tau.B = \rho.B$ and $\forall \tau, \rho \in \mathcal{S} \setminus \mathcal{C} / a \in \tau\ a \in \rho \Rightarrow \tau.B \neq \rho.B$

If those conditions hold, we say that there is an *association rule* with *support* s and *confidence* c in $\mathcal{U}$

The notation $A \mapsto B\ (s, c)$ will be used to denote this.

Note that A and B can be composite attributes and the definition still holds Similarly, the domains of A and B can be unitary

Example 1 Use of confidence and support to find rules See Table 6 1

The rule

"Symptom A = fever $\mapsto$ Disease X=present " has support 4 and confidence 0 75 The support set is $\{7, 8, 9, 10\}$ and the confidence set is $\{8, 9, 10\}$

The rule

"Symptom A = fever ∧ Symptom B= sore throat ↦ Disease X=present"

has support 3 and confidence 1

The rule

"Symptom B = no sore throat ↔ Disease X=present"

has support 7 and confidence 0 571

Theorem 1 $A \mapsto B$ *if and only if* $A \mapsto B$ $(\,|\,\mathcal{D}(\mathcal{U})\,|\;\; 1\,0\,)$

6.1.3 Association Rules in Decision Trees

In [4 pp 38 48] I have demostrated the relationship between functional dependencies and decision trees Those theorems establish this relationship

Notation Let $\mathcal{D}_t(A)$ a decision tree which classifies A i e., A is the target (goal) attribute for the classification Any feature or function x of $\mathcal{D}_t(A)$ is denoted $\mathcal{D}_t(A).x$ Height, size are features of a decision tree A function $subtree(i, j)$ denotes the subtree j at level i for all levels i and subtrees j of a decision tree

Theorem 2 *Let A a simple attribute*

$A \mapsto B \Leftrightarrow \exists \mathcal{D}_t(B) / \; \mathcal{D}_t(B)\ root = A \wedge \mathcal{D}_t(B)\ height = 1$

Theorem 3 *Let D a compound determinant attribute* $D = \{A_1, A_2 \;\; \dots, A_n\}$

$\Leftrightarrow \exists B \wedge \mathcal{D}_t(B) / \; \forall j, \mathcal{D}_t(B)\ subtree(i, j)\ root = A_i \wedge \mathcal{D}_t(B).height = n$

Theorem 4 *The height of the smallest decision tree is less or equal to the number of attributes of the shortest record key*

Theorem 2 guarantees that there is a tree if there is a simple functional dependency of the goal attribute

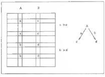


Figure 6.1 Illustration Theorem 2

Theorem 3 extends the result to composite dependencies of the goal attribute and indicates the kind of decision tree that is related to.

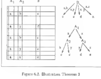


Figure 6.2. Illustration Theorem 3

Theorem 4 is just a corollary of the previous theorems and limits the height of the decision tree in the presence of record keys

Theorem 2 can be extended to association rules in general:

Theorem 5 Let A a simple attribute $c \in [0,1], s \in \mathcal{N}$

$A \mapsto B(s,c) \Leftrightarrow \exists \mathcal{D}_1(B) / \mathcal{D}_1(B).root = A \wedge \mathcal{D}_1(B).height = 1$ and $\exists \mathcal{R} \subset \mathcal{D}(\mathcal{A})/$

$$c \leq \sum_{a \in \mathcal{R}} \mathcal{C}(L(\{A = a\}))P(A = a) \tag{6.1}$$

$$s \leq \sum_{a \in \mathcal{R}} | L(\{A = a\}) | \tag{6.2}$$

Proof: $A \mapsto B(s,c) \Leftrightarrow$ From Theorem 2 $\mathcal{D}_1(B).root = A \wedge \mathcal{D}_1(B).height = 1$ Let $\mathcal{S}, \mathcal{C}$ and $\mathcal{R}$ as in the definition of the association rule. A and B coincide in every tuple in $\mathcal{C}$. Let c_a the number of tuples of $\mathcal{C}$ for value a of B, and let s_a the number of tuples of $\mathcal{S}$

Then, $| L(\{A = a\}) | = s_a$ and $\mathcal{C}(L(\{A = a\})) = \frac{c_a}{s_a}$ $P(A = a) = \frac{s_a}{|\mathcal{S}|}$

Leftrightarrow

$$\sum_{a \in \mathcal{R}} \mathcal{C}(L(\{A = a\}))P(A = a) = \sum_{a \in \mathcal{R}} \frac{c_a}{| \mathcal{S} |} = \frac{| \mathcal{C} |}{| \mathcal{S} |} \geq c \tag{6.3}$$

$$\sum_{a \in \mathcal{R}} | L(\{A = a\}) | = \sum_{a \in \mathcal{R}} s_a = | \mathcal{S} | \geq s \tag{6.4}$$

The following theorem extends the result

Theorem 6 A a composite determinant attribute $A = \{A_1, A_2, ..., A_n\}$ $c \in [0,1]$ $s \in \mathcal{N}$

$A \mapsto B(s,c)/ \Leftrightarrow \exists B \wedge \mathcal{D}_1(B)/$

$\forall j$ $\mathcal{D}_1(B).subtree(i,j).root = A_i \wedge \mathcal{D}_1(B).height = n$

and $\exists \mathcal{R} \subset \mathcal{D}(A)/$

$$\leq \sum_{a \in \mathcal{R}} C(L(\{A = a\}))P(A = a) \qquad (6.5)$$

$$s \leq \sum_{a \in \mathcal{R}} | L(\{A = a\}) | \qquad (6.6)$$

Note that if $a = \{a_1, a_2, \ldots, a_n\}$

$$P(A = a) = \prod_{i=1}^{i=n} P(A_i = a_i)$$

Proof: The attribute A can be considered as a single attribute with value $a = \{a_1, a_2, \ldots, a_n\}$ in every tuple. By previous theorem, equations 6.1 and 6.2 coincide with equations 6.5 and 6.6. The decision tree has height one and as root the composite attribute A. Then we can separate the attribute A for each A_i and the decision tree can be transformed to a decision tree with height n where $subtree(i, j) = A_i$ for all j.

6.2 Handling Many Valued Attributes

It has been shown (see chapter 5.) that attribute selection criteria tend to favor many valued attributes. The intuition behind this is that for small sets, an attribute with many values behaves almost as a primary key attribute and hence its determination of the class attribute is 100%. Many valued attributes affect the resulting set of derived rules since they are not relevant to the class determination, like the patient identification in a data set of diseases. Continuous attributes are special cases of many valued attributes (every continuous attribute is always represented by a very long sequence of discrete values). A concern exists in developing techniques for attribute selection that are not greatly influenced by the attribute cardinality such as the IDL system of Van de Velde [10] or techniques that can split up the attribute range to minimize the number of branches in the decision tree such as the gini index [8] or the Kolmogorov Smirnoff distance by two classes [13] recently improved by Utgoff and Clouse as a selection measure for decision tree induction [46]

The determination measure is not completely free from being affected by many valued attributes My concern has therefore been to incorporate in decision trees a way to decrease the range of the many valued attribute but at the same time increase or keep its determination – which is not always possible This means that each branch of the decision tree will be labeled with a range (even unitary ones) that represent the set of values. Note that attributes are not recoded, but that a range is just used as a label of the respective branch then preserving the original symbology of the user

This range compression technique –grouping together values of the attribute that maximise the determination measure or any other measure was implemented as a way to

1 Reduce the actual range of the attribute (mostly numerical attributes)

2 Allow comparison with other systems which are based on range splitting

3 Reduce the size of the resultant decision tree (less bulky) and therefore allow us to get a more compact tree and a set of derived rules.

6.2.1 The Best Split Partition Algorithm

Let $C(c\ j)$ be the class count distribution for class c and value j of a certain attribute. Then, the total number of cases N is given by $N = \sum_{c,j} C(c\ j)$

Let $M(j\ k)$ be any positive measure over the values j to k.

Let $\Pi_r = \{j_1, j_2 \ldots j_r\}$ a set of partition points over the set of values $v_1 \ldots v_n$ with $j_p \leq j_q$

if $p < q$ and j_1 [illegible]

Let $p(s,k)$ be the probability of range v_s [illegible]

$$p(s\ k) = \frac{1}{N}\sum_{l=s}^{j=k}\sum_{c} C(c\ j) \tag{6.7}$$

Let $M(\Pi_r)$ be the average measure over Π_r:

$$M(\Pi_r) = \sum_{i=1}^{i=r/2} p(j_{2i-1}, j_{2i}) M(j_{2i-1}, j_{2i}) \tag{6.8}$$

Definition 5 Π_r *is an optimum partition if it maximizes the value of* $M(\Pi_r)$ *with a minimum number of intervals i e , if there is another partition with the same value of M then it has more intervals*

Theorem 7 *If* Π_r *is optimum and* $\Pi_r = \Pi_s^1 \Pi_t^2$ *(a concatenation of two subinterval partitions) then*

$$M(\Pi_r) = p(\Pi_s^1) M(\Pi_s^1) + p(\Pi_t^2) M(\Pi_t^2) \tag{6.9}$$

where $p(\Pi_q) = p(j_1, j_q)$

The previous theorem says that an optimum partition is composed of optimum partitions of each subinterval This is useful for visualizing the following algorithm to get the optimum partition

```
Best Split algorithm
Best Split(integer I, integer N):
s0      Max= M([v(I) v(N)])  //the complete range
        If I=N return Max;
s1      For P=1 to  N 1 do
```

```
            h1 = Best Split(1,P)
            h2 = Best Split(P+1 N)
            M = p(1,P)*h1 + p(p+1 N)*h2
s1 1          If M_i Max then
                best = p
                Max = M,
         end
s2 return  Max
```

Correctness of the Best Split Partition

Theorem 8 The Best Split algorithm finds the optimum partition

Proof By induction on the number of ranges in the partition found for the Best Split algorithm, say Π_m^B

Basic case m=1 If Π_1^B is not optimum assume that Π_r is optimum ($r > 1$) then $M(\Pi_r) \geq M([v_1, v_n]) = M(\Pi_1^B)$ but steps s1 and s1 1 together with theorem 7 guarantee that the first range of Π_r must be found, so r must be 1

Induction Hypothesis Π_i^B is optimum for $i < m$

To show that Π_m^B is optimum assume that Π_r is an optimum partition First Π_m^B can be seen as the concatenation of the first two partitions $\{j_1 \ldots j_p\}$ and $\{j_{p+1}, \ldots, j_n\}$ where p is the maximal point found in step s1 Then each of those partitions is optimum for the respective subinterval by hypothesis

Assume $\Pi_r = \{o_1, o_2, \ldots, o_r\}$ then, we have two cases

$o_2 < j_p$ then Best Split must have detected it in step s1 1 before finding j_p since for theorem 7 $[o_1, o_2]$ and $[o_3, \ldots o_r]$ are optimum subpartitions and with maximum value

$o_2 > j_p$ then by a similar argument Best Split must have detected o_2 after finding j_p

Example 2 *Using Best Split to reduce the range according to the class determination.* Assume we have two classes The following table is the attribute value distribution for each class

class	Values			
	1	2	3	4
+	2	1	0	1
	1	0	2	1

```
Analyzing first partition of 1,2,3,4

[1] [2 3,4]  1   0 5 (3)

    Analyzing optimum for 2,3 4

    2 3 4  2   1 0 (1)

        Analyzing optimum for 3,4

        3 4   3   1 0 (2)

            4   0 0 (2)

              [3],[4]   1/4( 1 0*2+ 0 0*2) = 0.5

            [3,4]   (1  1/3)= 0 66 (4) (a)

            Optimum for 3 4 is [3,4] with value 0.66

        Evaluating first partition of 2 3,4

        [2] [3 4]  = 1/5 ( 1 + 4 * 0 66) = 0 73 (max)


        Analyzing optimum for 2,3

        2 3   2   1 0 (1)
```

3 : 1 0 (2)

[2],[3] : 1/3(1*1 6+ 2*1 0) = 1 0 (3) (b)

[2,3] (1 1/2)= 0 5 (3)

Optimum for 2,3 [2],[3]

4 : 0 0 (2)

[2],[3],[4] = 1/5 (3*1.0 + 2 * 0 0) = 0.60

Then the optimum for 2,3 4 is: [2],[3 4] with value

1/8(3 *0 5 + 5 * 0 73) = 0 644

Analyzing second partition of [1,2,3 4]

[1,2] [3 4] 1,2 1 0 5 (3)

2 : 1 0 (1)

[1],[2] 1/4 (3*0 5+ 1*1 0) = 0 625 (4)

[1,2] 1 1/3 = 0 666 (4)

[3 4] 0 666 (See (a) above)

1/8(4 *0 666 + 4 * 0.666) = 0 666 (optimum)

Analyzing third partition of [1,2 3,4[

[1 2,3] [4] 1 2,3 : 1 0 5 (3)

2 3 [2] [3] 1 0 (3) (See (b) above)

[1] [2],[3] 1/6(3*0.5+ 3 *1 0)= 0 750 (6)

1 : 0 8 (2)

1/8(8*0 75 + 2*8 0) = 0 562

Best option: [1 2],[3 4] with value 0.668

6.2.2 The Range Compression Algorithm

The Best Split algorithm is useful to make range compression when it is needed. I have implemented an approximation approach that uses the measure of the left and right subintervals instead of the optimum partition of each subinterval to choose the partition point or split point. This saves time in the computation of range compression but it does not guarantee an optimum range compression.

The objective of this algorithm is to maximize the measure of an attribute whose set of values and class count (frequency) distribution is given

Inputs Attribute, Number of Classes (Nclasses) list of attribute values and Class Counts ($f(v_i)$)

Outputs A list of range values which optimize the average measure (class determination) of the attribute

Range Compression Algorithm:

```
s0    [Traverse the value list and join consecutive values
        with the same class i.e. if $class(v_i) = class(v_{i+1})$ then
        $v_i$ and $v_{i+1}$ are in the same range ]
        Group values with same class into ranges
s1    [ Reorganize the ranges by recursively looking for the
        best two-way spliting (with large determination)
        for the accumulated class frequencies ]
        Get accumulated frequencies $F(r_i)$ where
        $F(r_1) = f(r_1)$ and $F(r_{i+1}) = F(r_i) + f(r_{i+1})$
s2    [ The best partition point is that that maximizes the
```

```
        average measure of the attribute. The maximum average
        measure must be larger than the actual attribute measure
        for the current value ranges
        Two lists are needed  one for accumulated class counts
        until certain value and other to hold the accumulated
        class frequencies for the remaining values
        If $L(r_i)$ is the accumulated frequency until range
        or value i and $R(r_i)$ is the accumulated frequency
        from range i to range n  thus
        $split = \max\{P(r_1 \ldots r_i) * M(L(r_i)) + P(r_{i+1} \ldots r_n) * M(R(r_i))\}$
        where P is the relative probability and M is the measure used
        When a partition point for the range is found  the remaining
        partition points are found by splitting the values to the left
        and to right of the partition point (if this is possible) ]
        Get the best partition points (if any) for the set of values (splits)
4     Join all the ranges accordingly to the Splits list
        Keep the last accumulated frequency of every split like the actual Class
        Counts (frequency) for that range
```

Example 3 *Reducing the range according to the class determination.*

Assume we have two classes The following table is the attribute value distribution for each class:

class	*Values*								
	1	2	3	4	5	6	7	8	9
+	6	7	4	6	5	8	2	5	0
	6	0	3	4	0	0	3	6	6

Step 10 Values 5 and 6 are merged in one range since they determine the same class

Class	Values							
	[1]	[2]	[3]	[4]	[5 6]	[7]	[8]	[9]
+	0	7	4	6	13	2	5	0
	6	0	3	4	0	3	0	8

The average determination for the table above is

$1 * (12 * 0.0 + 7 * 1.0 + 7 * 0.25 + 10 * 0.33 + 13 * 1.0 + 5 * 0.333 + 11 * 0.17 + 6 * 1.0) = 0.49$

Step #1 Finding splits

Table of accumulated frequencies by value Left and Right lists on every value are indicated by columns labels L and R.

	[1]		[2]		[3]		[4]		[5,6]		[7]		[8]		[9]	
	L	R	L	R	L	R	L	R	L	R	L	R	L	R	L	R
+	0	37	10	30	17	30	00	30	00	7	33	3	43	0	40	0
	6	20	6	22	6	18	13	15	11	11	16	16	30	0	28	0
Det	0.0	0.41	0.14	0.27	0.47	0.07	0.43	0.23	0.34	0.18	0.18	0.18	0.45	1.0	0.35	
Avg.	0.26		0.24		0.36		0.34		0.61		0.18		0.33		0.30	

Therefore the first split is in range [5,6] since 0 610 is maximum and greater than the original determination of 0 490

Similarly, there are no splits from 1 to [5,6] (calculations not shown) while there are splits in [7] and [9] After joining all those ranges the final ranges will be [1,6], [7] and [8 9]

6.2.3 Range Compression Experiments

As an example of what range compression can do an artificial database with 10000 cases was generated for two classes, with attributes in the range of 1 to 1000 A sample of 2000 rows was used to generated two decision trees, one without and one with range compression

Then the decision trees were used to extract the maximum support conjunctive rule (MSC rule) given for the tree This is the branch of the tree with have a larger set associated to each leaf

The terminology is the same used in chapter 5 on page 69 In the first case, the Initial

Table 6.2 Exp 5 Criterion Determination

Set	Sample size	Initial error	N I	% Ex Sd	Final error	Final S Size	Tree Size	Tree Ht	Tree Leaves	Tree Nodes	Tree Root	Rules Mean	Ext Tt
1(wo	3000	0.72	1	0	0.72	3000	[illegible]	2	1806	3601	A1	0.667	6
1(w	3000	2.61	1	0	1.44	1880	[illegible]	2	470	1163	A3	0.561	4

error was due to 67% soft cases since the long range of attributes was not considered in the tree In the range compression case the soft error was reduced to 35%, and hence the reduction in the initial error to 51% In addition the first decision tree leads to a MSC rule

If A1=302 then class = 1 (12 1 0)

and the second tree leads to a MSC rule:

If A1 > 833 then class =0 (23, 1 0)

Note how the support is increased from 12 to 23 and the rule is generalized as well However, both rules are still useful since each one describes a different class

In a separate test, a 100000 case database for two classes was generated Each attribute had 128 potential values. The trees were derived as before, with and without range compression, and the results compared

Table 6.3 Exp 5 Criterion Determination

Set	Sample size	Initial error	N I	% Ex Sd	Final error	Final S Size	Tree Size	Tree Ht	Tree Leaves	Tree Nodes	Tree Root	Rules Mean	Ext Tt
1(wo	10000	0.562	1	0	0.562	10000	[illegible]	1	3024	3000	A6	0.543	1001
1(w	10000	0.562	1	0	0.562	10000	[illegible]	1	3000	5030	A9	0.543	882

The external trees for both cases were similar in shape to previous cases 4 node trees with 3 leaves on average The maximum support rule derived for the first tree was

$\boxed{\text{If A0 is in [102 102] then class= 0 (21, 1 0)}}$

There were 0 532 % soft error cases for the simple tree, for a final net error of 13 % Mean while, there were 0 342 % soft error cases for the tree with range compression, for a final net error of 28 % The maximum support rule derived for this tree was

$\boxed{\text{If A0 is in [21 41] then Class= 0 (122, 1 0)}}$

Both trees had the same root with the same measure, which means the range compression didn't help for the root selection because of the random nature of the class assignment when the data base was generated However, the tree with range compression was smaller i e ,more of it was able to fit in memory and the MSC rule was more meaningful (larger support) for the range compression tree than for the simple tree

CHAPTER 7
COMPARISON WITH OTHER SYSTEMS

7.1 Comparison with Decision Tree Classifiers Systems

There are several approaches to solve the classification problem in Knowledge Discovery. Among others, the ITI system from Utgoff's [45] is a complete system to induce trees incrementally. However, the trees are kept memory resident, and hence the system is not useful for large databases, even though it implements almost all features described here. At the other extreme is the SLIQ system, which derives decision trees for large amount of data and hence keeps most of the data off line. Both systems represent the latest in tree construction. As SLIQ is an important approach to decision tree construction in data mining for very large data bases I compare approach described here with the SLIQ approach. See chapter 2 for a description of the SLIQ approach.

7.1.1 Analysis of SLIQ

1. SLIQ requires at most two passes over the data base for every level of the decision tree.

2. SLIQ builds a classifier (derives an accurate decision tree comparable to the standard decision tree algorithms) but it doesn't apply any incremental approach. Actually, it is not clear how an incremental approach can be integrated with their algorithm.

3. SLIQ is adequate for splitting of numerical attributes since the decision in every node is implemented as an expression $A <= v$. This reduces the number of branches when

many values attributes are present (e g a floating point number) This is application dependent One may recode a many values attribute to reduce its range which has the advantage of being user dependent and not class dependent A class dependent partition as implemented by SLIQ could not satisfy the user point of view Another approach is to prune the final tree and merge branches that lead to the same class.

7.1.2 General Comparison with a Decision Tree Based Approach

The following is a list of the SLIQ features versus a decision tree approach as proposed in this work

1 SLIQ derives a classifier (a decision tree) equivalent to other entropy based algorithms (C4 5) It uses the gini index as the criterion for attribute selection

   Our classifier is a decision tree based on entropy or determination which is as good as any other entropy based mechanism We can derive the same decision tree if the gini index is used despite the performance differences, and so the accuracy as a classifier is the same

2. SLIQ is scalable, thanks to the use of external storage for long range attributes and to the use of inverted lists stored on disk

   I have already implemented a splitting approach (range compression algorithms) which can be made binary or n ary (hence more general) and a method to store subtrees in disk which makes the system scalable in the same sense

3 SLIQ requires at most one pass over the data (two times the number of I/O's due to the space duplication (see below) when created inverted lists for attributes) per level of the decision tree It assumes the class list can be kept memory resident

Our tree derivation, which is essentially the SLIQ system takes at most one pass per level of the decision tree (one time the number of I/O's) if we keep the last level class counts memory resident which is the same assumption that SLIQ makes about the class list

4 Rules extracted from a decision tree derived from SLIQ will be based on the binary splits of the attributes ($A < v$ or $A > v$) and hence they will be longer, more general and less meaningful than rules based on a decision tree based on interval splitting for attributes (our system)

5 SLIQ requires two times more space than the original database (since columns are kept as separated list with indices attached) while the basic decision tree algorithm requires at most one time more space which can be reduced to a minimum using an key index or simply partitioning the data base This is again particularly important for very large databases

6 The Decision Tree Approach can be extended to a distributed approach (see chapter 4) It is not clear how SLIQ can be extended to a distributed approach (it was not elaborated by Metha, Agrawal and Rissanen [26])

7 SLIQ was not designed to be incremental. Meanwhile our decision tree construction can be incremental using tree reorganization techniques

7.1.3 Memory Comparison

The following assumptions have been made for comparing both systems on memory requirements

1 Both systems will be used for tree derivation but not for incremental purposes

2. SLIQ will keep the Class List in main memory. Our system will keep the last level class counts of the tree or trees in memory and only those since we are concerned with tree derivation without incremental features.

3. We use the gini index as the method to select the best attribute (to make the comparison compatible).

4. The database consists only of numerical features. This can be relaxed if we modify SLIQ to apply the gini index to categorical attributes.

Thus, SLIQ will require $N * 8$ bytes of main memory where N is the number of tuples of the database. If A is the number of attributes, V the average number of values, C the number of classes and H the height of the decision tree, then we will have

$$2^{H}(A-H) * V * C * 16 \tag{7.1}$$

bytes of memory required for our algorithm (the TIVLD algorithm). Note that each counter requires 16 bytes, for keeping track of the attribute, value and class. There are $(A-H)*V*C$ counters in every node at level H.

Then, SLIQ will require more memory if

$$2^{H+1}(A-H) * V * C < N \tag{7.2}$$

Equation 7.2 can be used to select one or other algorithm based on memory requirements. Note that the height H is unknown before hand and it must be estimated. If the highest value for height A is chosen then we might use SLIQ when in practice a TIVLD (Tree Induction for Very Large Databases) algorithm will perform better if the decision tree is small. In any case, SLIQ will require two times the number of I/O's and an initial presorting phase

7.1.4 Conclusions

The results obtained by Mehta et al [26] show that SLIQ can be used effectively for large data sets with linear scalability The comparisons shown in that paper with other systems seem unfair since they were designed with different goals in mind to keep data tuples inside the tree and minimize the amount of central memory required, without caring about the number of passes over the data and so on

The theoretical comparison made here shows that while keeping the same goals the traditional tree derivation algorithm can be modified to get adequate performance for very large databases

7.2 Comparison with Systems to Derive Association Rules

The definition of association rule given in chapter 6 is general for *standard databases* a database consisting of n tuples (rows) and attributes (columns); where there is no limitation to the values that each attribute may have (as long as they are normalized, at least in 1NF) This implicitly assumes that you might have a relational table as your standard database. Since the term *relational database* includes several normalized tables it usually means more than a single no completely normalized table I use the first term

Agrawal et al terms the *transactions database* a collection of transactions where each transaction consist of a collection of items [1]

The original definition introduced by Agrawal for association rules between items states that two subsets of items X Y are associated if there are transactions that contain both X and Y In addition, it is assumed that an implication of some sort exist from X to Y and is denoted $X \Rightarrow Y$ The *support* and *confidence* were defined in terms of the number of transactions with X Y and $X \cup Y$ The *support* in this sense is the maximum number of transactions containing X or containing Y The *confidence* is the ratio $\frac{support(X \cup Y)}{support(X)}$ Note that our definition of support of the association rule is the support of the antecedent. According to Agrawal [1], the support of a rule is constant and doesn t depend on the support of the antecedent

7.2.1 Standard Databases to Items Databases

My formal definition of *association rule* is more general and subsumes the above definition Consider every item description as a column in a new database and each transaction mapped to a row where there will be 1 if the item described in the respective column is included in the transaction, 0 otherwise This database will be called the *items database*)

Thus, if an association rule exists in the sense of Agrawal in [1], between two sets X and Y than let S be the set where X=1 (i e., where every item column is 1) and C be the set where both X=1 and Y=1 Thus, we have $X \mapsto Y$ (s c) with s the cardinality of S and $c = \frac{|C|}{|S|}$

Since systems for extracting association rules are oriented to *items databases*, we need a way to map a *standard database* to those in such a way that we can compare decision tree algorithms to association rules ones

The translation is easy Consider an "item" every value of every attribute Thus, each tuple will be mapped to a "transaction" that contains all values of all attributes in the tuple If a value is missing simply don t include the "item" Note that the size of the itemset in a "transaction" will be lower than or equal to the number of original attributes

7.2.2 Global Features Comparison

The transformation of the data base described in the previous section allow us to start a comparison between the decision tree approaches (DT) and association rules approaches (AR)

- *Portion of the rule*: Actual AR algorithms are able to derive all association rules from the data base with a minimum specified support and a minimum specified confidence. The consequent of the need not be simple as it can be a composition of several items (or attributes)

  DT classical algorithms need to be parallelised to get a similar result, since every goal attribute (simple or compound) represents a potential decision tree. However, since the goal attribute has several values and each value is an "item" in the items database the DT algorithms are extracting several rules simultaneously

  Additionally, if $A \mapsto B(s1, c1)$ and $A \mapsto C(s2, c2)$ then

  $A \mapsto B + C, (\max(s1, s2), \min(c1, c2))$ + denotes concatenation.

  In this case, only single consequent parts are necessary, this diminishes the amount of parallelism needed.

- *Redundant work*: On the other hand, AR algorithms will derive the composite rule above whenever the first two rules satisfy the thresholds required, creating some redundant work. Note that from the point of view of the items database, there are no ways to differentiate between the above rules, even with rules whose consequent "item" is related to the same attribute in the original data base.

- *Range compression and recodification*: It is easy to incorporate range compression to DT approaches which allow us to create rules with larger support. It is possible to recode attribute values to reduce their range. Recodification can be done before creating the *items database* but range compression is just a feature of DT approaches and can not be applied with an AR approach.

- *Criteria*: Entropy determination and the gini index are criteria that can be applied to extract different sets of more general association rules which are not present in a simple standard to item transformation.

- *Attribute priorities*: The criterion used to extract attributes as roots of the decision trees allows us to process the attributes and rank them according to their significance. This information is lost with the transformation to an *items database*.

- *Incremental approaches*: There have not been proposals to implement an incremental approach to extract association rules. In all AR algorithms the whole database is processed. However A. Savasere minimizes the number of passes over the database [37].

Based on the above observations, DT approaches offer several advantages that can't be achieved with AR approaches. However, assuming that one does not need such advantages and is interested only the sample rule extraction, I present a theoretical comparison of the DT algorithms with the actual AR algorithms below.

7.2.3 Approach Using a Classical Decision Tree Algorithm

In order to use decision trees we have to define target attributes. Since we don't know beforehand which attributes are important for the application, a general approach is to derive the decision tree for every attribute (In a real application, people will be interested in specific attributes, except if they want every possible relationship) Then, we have A decision tree derivations, where A is the number of attributes In order to get the same confidence and support numbers of the association rules algorithm above we have to test the decision tree against the whole transaction data base Thus, the complexity of the decision tree approach will be CDT= A *(Passes to build the tree + one pass) since we have A decision trees The Passes to build the tree will be proportional to the number of attributes (there will be A passes over a subset of the transaction data base) So, Passes to build the tree= A*S where S is the proportional size ($0 < S \leq 1$) of any subset of the data base and $CDT = A(A * S + 1) = A + S * A^2$

In general, this precludes the use of a classical decision tree algorithm, because the association rules algorithm will make at most A passes over the data and CDT will be always higher than A (it does not matter how small we choose S)

7.2.4 Approach with the Multiple Goal Decision Tree algorithm

The MGDT algorithm described in chapter 3 derives the decision tree for a set or all attributes in the database. It derives the trees breadth first (different from our Revisited Derivation algorithms) and reads the data base once at each level of all trees Thus, we extract all trees with A passes over the database

This is a comparison of the association rules system (Apriori algorithm) (AR) and our approach(MGTD)

If A is the number of attributes the complexity of the AR system in terms of passes over the database is. $CAR = O(A)$ and the complexity of the MGTD system is $CDT = O(A * S + 1)$ where $0 < S <= 1$ if the A decision trees are derived in parallel

In this case the decision tree approach will be better in general if $S < 1 - 1/A$, which will be true almost for every A

Even a direct tree derivation (without induction) approach will be equivalent in both cases with the same complexity $CDT = O(A)$ since $S = 1$, and the test phase is not needed

7.2.5 Summary and Conclusions

The MGDT algorithm offers an additional advantage it is possible to speed up the process using the user confidence and support to stop the construction of subtrees and limiting the amount of necessary work The Apriori algorithm and similar ones need to derive the complete itemset and then calculate the association rule with subsets of the itemset and the user confidence. It could be the case that none of the subsets satisfy the user confidence for the rule It is not possible to use the user confidence before the whole itemset is defined

CHAPTER 8
CONCLUSION AND FUTURE WORK

8.1 Conclusions

The decision tree approach is important since decision trees solve the classification problem and I have shown that can be effectively used for rule extraction. Then their application to Knowledge Discovery is imminent. Their application to very large databases (distributed or otherwise) requires algorithms that minimize the number of passes over the data while preserving the accuracy of classification and the confidence/support of the potential rules. Ours is one of the first attempts to propose and use decision trees for discovering quantitative rules in very large and distributed data bases. This area is actually of primary concern in Knowledge Discovery and Data Mining. See for example [43] [18].

In relation to our model of Knowledge Discovery described in the introduction, I can summarize my contributions in the different components of the model. Besides features that must be enhanced in the model are mentioned if one likes to use our approach of decision tree construction for Knowledge Discovery.

- *Interface*. Besides the experiments, an application of the decision tree approach was developed for a potential large database for ECMO (ExtraCorporeal Membrane Oxygenation) data. This database was created by Drummond [11] and it consists of a large clinical data collected minute-by-minute of infants who are critically ill. The data was reduced for the purposes of this application to a small data set of relevant cases and the results are not included here since most of the features proposed in this work were not applicable however the results were meaningful for the expert

The use of decision trees in the experiments and in the practical application allows us to visualize several operators that must be implemented in a Data Mining Manipulation Language (DMML) in order to effectively interface with an existent database:

- *Indexing* It is evident from the Multiple Goal algorithm in chapter 4 that fragmenting the data base is not useful in this case since we have to locate each instance in different subtrees So local subsets in each subtree must be kept as pointers to the original database The DMML should provide this capability to the Decision Tree Based System.

- *Selecting* In real applications just a few values of the goal attributes can be of concern for the end user Developing the decision trees or rules for all of them is not required or important The DMML must be able to provide only the required rows of the database. Database subschemas , Views or SQL statements can achieve this requirement but still this is not transparent enough for the Data Mining Tool Designer

- *Attribute joining* Decision Tree algorithms can be complicated enough when only a goal attribute is used If several attributes must be considered as a unique goal attribute the algorithm does not change but the interface must suffer a lot of changes The DMML must provide a way to retrieve as a unique value the joint value of several goal attributes

- *Aggregate Attributes:* Similar to the previous requirement, real dependencies can be captured only in aggregate attributes A way to combine and summarize previous row values (for time-series databases) , and create alternate attributes must be provided

- *Irrelevant Attributes* When there are attributes that are not required to be processed the DMML must help in this matter Although, SQL statements are able to provide this the interface must be such that all requirements mentioned can be met in a few operations
- *Keep primary keys* For data analysis and classification the user might need to use/verify the local subset Keeping the primary keys for local subsets must be important in some applications
- *Pre selected attributes* In the same way that some attributes can be considered irrelevant some of them can be considered relevant and must be included in early stages (as roots) of the decision tree extracted or well for simple data analysis
- *Frequency calculations* Most of the decision tree derivation time is time expended in frequency calculations If the Database System has effective and efficient ways to do the same work it must be a way to implement better algorithms for decision tree derivation
- *Recoding* Many values attributes must be filtered and grouped in the fly [illegible], when they are read to lower the number of classes when they are used as target attributes Thus no real changes are made to the database

- *Focus component.* Selection statements in the interface as well as keeping/ discarding attributes are ways to focus in the necessary data. Additionally in the early stages of decision tree derivation, attributes with low certainty measures can be discarded from additional computations Minimum threshold values can be provided to do this

- *Pattern Extraction component:* All the improvements on decision tree construction mentioned in chapter 4 can be included here Among others are the determination criteria, the range compression algorithms, the distributed algorithms, incremental approaches and induction in large databases.
- *Evaluation component:* The greedy approach of the attribute selection in decision tree construction allows us to evaluate rules before they are completed The Determination measure is a useful tool in this sense as show in chapter 5 The Knowledge Discovery model suggests this evaluation as a final component in the process Decision Trees allow us to evaluate rules even before they are completely extracted.
- *Knowledge Representation.* As a last contribution, it must be noted that decision trees are able to represent rules in a very concise way The natural hierarchy of decision trees allows us to extend them to the most complicate types of rules which are of actual interest See [42] [20]

To conclude, I must quote Robert Grossman [17, pp 24]

"When faced with a high-dimensional attribute space tree-based techniques which in a greedy fashion split the data one attribute at a time are generally far superior to techniques which require examining some combination of attributes"

I expect the results of this thesis to be beneficial for tree derivation and for tree induction as well.

8.2 Future Work

A number of issues that are not fully addressed in this work are·

- Full implementation of the system The implementation I made for experimental purposes does not include all features mentioned earlier we need to refine the incremental approach and work more in the Multiple Goal part

- Analysis of the effects of the incremental approach with respect of the shape of the decision tree and in the final rules It is clear that the previous trees resemble the actual tree when incremental approaches are used It will be important to measure this resemblance in terms of the number of matching nodes, branches, height and so on Experiments with large data bases are important for this purpose

- Using decision trees for representing second order rules Propositional rules as the association rules defined here are based on first order logic It is of interest to extract high level rules

- Application to real very large databases We used synthetic data bases for the experiments, but the behavior of the decision trees algorithms in most uncontrolled environments is always of concern

- Use of DMML The Data Mining field is just emerging. Researchers are doing mostly file mining rather than database mining [22]. When DMML are available, it will be important to evaluate the performance of decision trees algorithms See [21]

- Improving on the range compression algorithm The implementation of the range compression does not include the Best Split algorithm described in chapter 6 It seems a Look up algorithm can be easily implemented since most subrange calculations are repeated in the algorithm

- Incorporate a way to make the selection criteria user dependent. Although, we have incorporated several criteria into the implementation, new criteria will require new programming. User dependent implementation of criteria can be better suited to specific environments.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, Usa, 1993.

[2] R. Agrawal and J. C. Shafer. Parallel mining of association rules. In *Proceedings of EDBT 96*, France, 1996.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining of association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago, Chile, 1994.

[4] J. R. Arguello. Decision trees: Tools of ai and data modeling. Master's thesis, University of Denver, 1988.

[5] J. R. Arguello. Toward building the best decision tree. In *Proceedings of First Rocky Mountain Symposium on Artificial Intelligence*, pages 187–198, Boulder, Colorado, 1986.

[6] J. R. Arguello and S. Chakravarthy. Distributed tree induction for knowledge discovery in very large distributed databases. In *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 1–8, Montreal, 1996.

[7] P. Benedict and L. Rendell. Data generation program/2 v1.0. Technical report, Inductive Learning Group, University of Illinois, Urbana, 1990.

[8] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and regression trees. Technical report, Wadsworth, Belmont, 1984.

[9] C. J. Date. *An Introduction to DataBase Systems*. Addison Wesley Publishing Company, Reading, Massachusetts, 1995.

[10] W. V. de Velde. Incremental induction of topologically minimal trees. In *Machine Learning: Proceedings of the Seventh International Conference*, pages 66–74, University of Texas, Austin, Texas, 1990.

[11] W H Drummond, J M Bosworth, D W Kays D L Sandler M R. Langham and C E Wood Activation of the renin angiotensin system during neonatal ecmo In *Pediatric Research*, volume 35, page 223a Society of Pediatric Research, 1994

[12] W J Frawley, G Piatetsky Shapiro, and C J Mathews *Knowledge Discovery in Databases, an Overview* pages 1–27 AAAI Press/The MIT Press, Massachusetts, 1991

[13] J H Friedman. A recursive partitioning decision rule for nonparametric classification. In *IEEE Transactions on Computers* volume C 26 pages 404–408 New York, 1977

[14] L M Fu *Neural Networks in computer intelligence.* Mc Graw Hill Inc , New York, 1994

[15] R M Goodman and P Smyth Decision tree design from a communication theory standpoint In *IEEE Transactions in Information Theory*, volume 34, pages 979–994, New York, 1988

[16] R M Goodman and P Smyth Decision tree design using information theory In *Knowledge Acquisition*, volume 1, pages 1 19 New York 1990

[17] R Grossman H Bodek and D Northcutt Early experience with a system for mining, estimating, and optimizing large collections of objects managed using an object wharehouse In *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery* pages 21–26 Montreal 1996

[18] R. Hamming *Coding and Information Theory*, pages 101–104 Prentice Hall, Englewood Cliffs N.J., 1980

[19] J Han Y Cai and N Cercone Data driven discovery of quantitative rules in relational databases. In *IEEE Transactions on Knowledge Discovery and Data Engineering*, volume 5 pages 29–40, New York 1993

[20] J Han and Y Fu Discovery of multiple-level association rules from large databases In *Proceedings of the 21st International Conference on Very Large Data Bases*, pages 420–431 Switzerland, 1995

[21] J Han Y Fu, W Wang, K Koperski and O Zaiane Dmql a data mining query language for relational databases In *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery* Montreal 1996

[22] T Imielinski From file mining to database mining In *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery* Montreal 1996

[23] Y Kodratoff and R Michalski *Machine Learning An Artifical Intelligence Approach*, volume III Morgan Kaufmann San Mateo California, 1990

[24] C J Mathews P K Chan and G Piatetsky Shapiro Systems for knowledge discovery in databases In *IEEE Transactions on Knowledge and Data Engineering* volume 5(6) pages 903–913, New York, 93

[25] M Mehta, R. Agrawal, and J Rissanen Mdl based decision tree pruning In *Proceedings of Int'l Conference on Knowledge Discovery and Data Mining (KDD-95)*, Montreal, Canada, 1995

[26] M Mehta, R. Agrawal and J Rissanen. Sliq A fast scalable classifier for data mining In *Proceedings of EDBT 96 France March 1996* France, 1996

[27] R. S Michalski, J G Carbonell, and T. M Mitchell *Machine Learning An Artificial Intelligence Approach*, volume I Tioga Publishing Company, Palo Alto, California, 1983

[28] S Mullender *Distributed Systems*. Addison Wesley, Reading, Massachusetts, 2 edition, 1993

[29] G Piatetsky-Shapiro and W J Frawley *Knowledge Discovery in Databases* AAAI Press/The MIT Press Massachussets, 1991

[30] J R Quinlan *Machine Learning An Artificial Intelligence Approach*, volume I, pages 463–488 Tioga Publishing Company Palo Alto California, 1983

[31] J R. Quinlan Induction of decision trees In *Machine Learning*, volume 1, pages 81 106 1986

[32] J R Quinlan An empirical comparison of genetic and decision tree classifiers In *Proceeding of the Fifth International Conference on Machine Learning* pages 135–141, 1988

[33] J R Quinlan Inferring decision trees using the minimum description length principle In *Information Computer*, volume 80 pages 227–248 1989

[34] J R Quinlan Probabilistic decision trees In *Machine Learning Proceedings of the Seventh International Conference* pages 90 97, University of Texas, Austin Texas, 1990 AAAI Press/The MIT Press

[35] J R. Quinlan Foreword In G Piatetsky Shapiro and W J Frawley editors *Knowledge Discovery in Databases an Overview*, pages ix–xiii AAAI Press/The MIT Press, 1991

[36] J R. Quinlan *C4 5 Programs for Machine Learning* Morgan Kaufman, San Mateo, California, 1993

[37] A Savasere E Omiecinski and S Navathe. An efficient algorithm for mining association rules in large databases In *Proceedings fo the 21st International Conference on Very large Data bases*, pages 432–444 Zurich Switzerland 1995

[38] J C Schlimmer A case study of incremental concept Induction In *Proceedings of AAAI*, pages 496–501, Philadelphia, 1986

[39] C E. Shannon. The mathematical theory of communication In *Bell Systems Technical Journal*, volume 27(3) pages 379–423 1948

[40] C E Shannon *The Mathematical Theory of Communication*, chapter 1, pages 7–26 The University of Illinois Press Urbana 1949

[41] P Smyth and R. M Goodman An information theoretic approach to rule induction from databases In *IEEE Transactions on Knowledge and Data Engineering* volume 4 (4), pages 301–316, New York. 1992

[42] R. Srikant and R. Agrawal. Mining generalized association rules In *Proceedings of the 21st International Conference on Very Large Data Bases*, pages 407–419, 1995

[43] R. Srikant and R. Agrawal Mining quantitative association rules in large relational tables In *ACM SIGMOD96 International Conference on Management of Data*, 1996

[44] Paul E Utgoff Incremental induction of decision trees In *Machine Learning*, volume 4, pages 161 186 1989

[45] Paul E Utgoff Decision tree induction based on efficient tree restructuring Technical Report 95-18 Department of Computer Science University of Massachusetts 1995

[46] Paul E Utgoff and Jeffery A Clouse A kolmogorov-smirnoff metric for decision tree induction Technical Report 96-3. Department of Computer Science University of Massachusetts 1996

BIOGRAPHICAL SKETCH

Jose R. Arguello received his Bachelor of Computer Science degree from the University of Costa Rica in 1976 and a License degree in computer science in 1979 from the same university. Since then he has worked as a professor at the University of Costa Rica. He was awarded a Master of Science degree from the University of Denver in 1986, after which he returned to the UCR where he was chairman of the Department of Computer Science from 1989 until 1992. He started his Ph.D. program in computer science in the Computer & Information Sciences & Engineering Department of the University of Florida, Gainesville, in Summer 1993 and is scheduled to graduate in Summer 1996. He is interested in artificial intelligence, data bases and computer networks, and he will continue his work at the University of Costa Rica.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate in scope and quality, as a dissertation for the degree of Doctor of Philosophy

Sharma Chakravarthy, Chair
Associate Professor of Computer and Information Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality as a dissertation for the degree of Doctor of Philosophy

Andrew Laine
Associate Professor of Computer and Information Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy

Richard Newman Wolfe
Assistant Professor of Computer and Information Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate in scope and quality, as a dissertation for the degree of Doctor of Philosophy

Li Min Fu
Associate Professor of Computer and Information Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Paul Avery

Paul Avery
Associate Professor of Physics

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

August 1996

Winfred M. Phillips
Dean, College of Engineering

Karen A. Holbrook
Dean, Graduate School